

FUNCTION-TRAIN: A CONTINUOUS ANALOGUE OF THE TENSOR-TRAIN DECOMPOSITION

ALEX GORODETSKY*, SERTAC KARAMAN*, AND YOUSSEF MARZOUK*

Abstract. We describe a new nonparametric function approximation framework based on a continuous extension of the tensor-train decomposition. The approximation, termed a function-train (FT), results in a tensor-train structure whose cores are univariate functions of each variable. An advantage of the FT over discrete approaches is that it produces an adaptive approximation of tensor fibers that is not tied to any tensorized discretization procedure. Furthermore, the representation of low-rank functions in FT format enables efficient continuous computation: we can add, multiply, integrate, and differentiate functions in polynomial time with respect to dimension. Our approach is in the spirit of other continuous computation packages such as Chebfun, and yields an algorithm which requires the computation of “continuous” matrix factorizations such as the LU and QR decompositions of vector-valued functions. Our contributions include creating a maxvol algorithm for finding the maximum volume submatrix of a matrix-valued function. We also introduce a maxvol cross-approximation algorithm to obtain skeleton decompositions of vector-valued functions, a cross-approximation algorithm for converting black-box functions into FT format, and a continuous rounding algorithm that re-approximates an FT by an FT of lower ranks. We demonstrate the benefits of our approach by integrating high dimensional and discontinuous functions. We also show how the algorithm performs on a variety of approximation tasks.

1. Introduction. Computing with functions is an emerging paradigm posing an alternative to the traditional *discretize-then-solve* [21] methodology used in many computational science applications. This paradigm shift has been enabled by the development of continuous extensions to common and widely used linear algebra techniques that make up the building blocks of virtually all computer simulations. For example, in [4, 24, 31, 32] an extension of Matlab, called Chebfun, has been created for dealing with functions of one and two variables. Chebfun enables functional factorizations that are continuous versions of LU, QR, and SVD decompositions, maximization and root finding, and solution methods for ordinary differential equations without discretization.

In this work, we present a methodology that enables the extension of *continuous computation* to higher dimensions. We propose this extension by representing high dimensional functions in a *compressed* form, and the degree of compression is tied to the *rank* of multidimensional functions. In particular, we draw on recent developments in tensor decompositions, mainly the tensor-train (TT) decomposition [23, 22], to create an analogous continuous decomposition called the function-train (FT) decomposition. In this framework, we will consider the FT rank of functions instead of the TT rank of arrays. While the notions of FT rank and TT rank are analogous, there exist other differences between the TT and the FT decompositions. The most striking difference is that the FT cores are matrix-valued functions rather than third order arrays. The matrix-valued functions represent set of univariate functions, and we follow Chebfun’s example by representing each univariate function in a basis of orthonormal polynomials or piecewise polynomials. This continuous framework provides several computational advantages over working with multidimensional arrays arising from high dimensional discretization procedures. We are able to add and multiply functions together without specifying matching discretizations, perform fast integration without specifying tensor product quadrature nodes and weights, and inject additional information about function regularity to reduce computational cost for approximation.

The algorithms we describe in this paper are built using continuous linear algebra tools, and we review these tools in Section 2.2. We begin by discussing the principal elements of continuous linear algebra: scalar-valued, vector-valued, and matrix-valued functions. We then discuss factorization techniques for representing these elements in low-rank format through functional QR, LU, and SVD decompositions. Then, in Section 3 we use these tools to develop a cross approximation for obtaining a skeleton decomposition of a vector-valued function. These factorization procedures and cross approximation algorithms provide the building blocks for the cross-approximation of a multi-

*Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA 02139, USA, {goroda,sertac,ymarz}@mit.edu.

dimensional function in FT format. In Section 4 we formally define the function-train and discuss the cross-approximation algorithm used to convert a black-box function into its FT representation. We also develop continuous version of the tensor-train rounding algorithm to reapproximate a FT by one with lower ranks, and the FT rounding procedure serves as the basis of a rank-adaptation algorithm. The concept map in Figure 1 displays how the various areas of continuous linear algebra will relate to the function-train decompositions and will help serve as a roadmap for this paper.

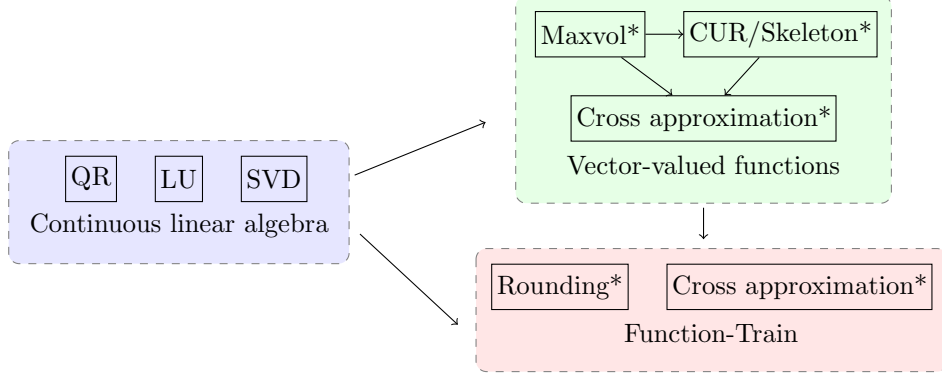


FIG. 1. Concept map relating the topics discussed in this paper. Arrows point from concepts that serve as building blocks to other concepts. Starred entries indicate areas of novel contributions.

In Section 5 we discuss how we can perform continuous computation using the FT representation. In particular, we show how one can add, multiply, integrate, and differentiate functions in FT format. All of these procedures can be performed in polynomial time with dimension, and we believe that these multilinear algebra algorithms can serve as the basis for performing continuous computation for a wide variety of applications. In Section 6 we discuss implementation details. We first focus on describing how the univariate functions that make up the FT cores can be represented in polynomial- or piecewise-polynomial bases. We then describe how representing the univariate functions in terms of basis functions allows us to perform efficient numerical integration. Finally, in Section 7 we demonstrate some numerical integration and approximation examples.

2. Preliminaries.

2.1. Notation. Let \mathbb{Z}^+ denote the set of positive integers and \mathbb{R} the set of reals. Let $\mathcal{X} \subset \mathbb{R}^d$ for $d \in \mathbb{Z}^+$ be the set defined as a tensor product of one-dimensional intervals $\mathcal{X} := [a_1, b_1] \times [a_2, b_2] \times \dots \times [a_d, b_d]$ with $b_i > a_i$ for $i = 1, \dots, d$. We can denote a scalar-valued function as a mapping $f : \mathcal{X} \rightarrow \mathbb{R}$, a vector-valued function as a mapping $F : \mathcal{X} \rightarrow \mathbb{R}^n$, and a matrix-valued function as a mapping $\mathcal{F} : \mathcal{X} \rightarrow \mathbb{R}^{n \times m}$ for $n, m \in \mathbb{Z}^+$. We can refer to an indexed output of F or \mathcal{F} through indices $i \in \{1, \dots, n\}$ and $j \in \{1, \dots, m\}$. Then we can index vector-valued and matrix-valued functions as $F[i] : \mathcal{X} \rightarrow \mathbb{R}$ and $F[i, j] : \mathcal{X} \rightarrow \mathbb{R}$, where both $F[i]$ and $\mathcal{F}[i, j]$ are scalar-valued functions.

Let $\mathcal{X}_{\leq k} = [a_1, b_1] \times [a_2, b_2] \times \dots \times [a_k, b_k]$ and $\mathcal{X}_{> k}$ be defined such that $\mathcal{X} = \mathcal{X}_{\leq k} \times \mathcal{X}_{> k}$. We can then refer to *separated* interpretations of scalar-valued functions f^k , vector-valued functions F^k , and matrix-valued functions \mathcal{F}^k as

$$(2.1) \quad f^k : \mathcal{X}_{\leq k} \times \mathcal{X}_{> k} \rightarrow \mathbb{R} \Rightarrow f^k(\{x_1, \dots, x_k\}, \{x_{k+1}, \dots, x_d\}) = f(x_1, \dots, x_d),$$

$$(2.2) \quad F^k : \mathcal{X}_{\leq k} \times \mathcal{X}_{> k} \rightarrow \mathbb{R}^n \Rightarrow F^k(\{x_1, \dots, x_k\}, \{x_{k+1}, \dots, x_d\}) = F(x_1, \dots, x_d),$$

$$(2.3) \quad \mathcal{F}^k : \mathcal{X}_{\leq k} \times \mathcal{X}_{> k} \rightarrow \mathbb{R}^{n \times m} \Rightarrow \mathcal{F}^k(\{x_1, \dots, x_k\}, \{x_{k+1}, \dots, x_d\}) = \mathcal{F}(x_1, \dots, x_d),$$

for $x_i \in [a_i, b_i]$ for $i = 1 \dots d$. Also, the shorthand $x_{\leq k}$ will denote $\{x_1, \dots, x_k\}$ and $x_{> k}$ will denote

$\{x_{k+1}, \dots, x_d\}$. Let us denote the following indexing of vector-valued functions

$$\begin{aligned} F^k[i](x_{\leq k}, \cdot) : \mathcal{X}_{>k} &\rightarrow \mathbb{R} \\ F^k(\cdot, x_{>k}) : \mathcal{X}_{\leq k} &\rightarrow \mathbb{R}^n \end{aligned}$$

Let bold faced lower case letters $\mathbf{a} \in \mathbb{R}^n$ denote a vector and bold face upper case $\mathbf{A} \in \mathbb{R}^{n \times m}$ define a matrix.

2.2. Continuous linear algebra. Recall that the approximation of a black-box tensor using the tensor-train decomposition algorithm requires performing a sequence of common numerical linear algebra routines such as LU, QR, and SVD decompositions. Our *continuous* framework will therefore require continuous equivalents of these decompositions. The continuous linear algebra factorizations which are required for the function-train decompositions must be performed for three elements of continuous linear algebra: scalar-valued functions, vector-valued functions, and matrix-valued functions. In this section, we describe how these three elements relate to each other, how they can be used in multiplication operations, and how we can factorize them.

The LU, QR, and SVD of vector-valued and multi-dimensional functions is of prime importance for practical linear algebra and serve as primary ingredients for any infinite dimensional numerical linear algebra package such as Chebfun [24]. They also serve as the building blocks for higher dimensional constructions. These decompositions, in the context of vector-valued functions has been previously discussed in [4]. Their computations often entail continuous analogues of common, discrete algorithms. For example, Householder triangularization may be used for the computation of the QR decomposition a vector-valued function [33]. While in [4] decompositions of one-dimensional functions was discussed, the extension of these algorithms to two-dimensional functions was provided in [32]. We will use these two-dimensional extensions in our algorithm, and we spend some time reviewing the literature.

2.2.1. Elements of continuous linear algebra. A vector-valued function is viewed as an array of scalar-valued functions. Suppose that we are considering scalar functions $f_i : \mathcal{X} \rightarrow \mathbb{R}$. Then a vector-valued function F consists of n functions

$$F = [f_1(x) \ f_2(x) \ \dots \ f_n(x)] \Leftrightarrow F[i](x) = f_i(x) \Leftrightarrow F^k[i](x_{\leq k}, x_{>k}) = f_i(x_{\leq k}, x_{>k})$$

There are multiple interpretations of vector-valued functions which one may find useful in different contexts.

- “ $\infty \times 1$ ” vector (used in Sections 2.2.3 for the LU decomposition and 2.2.4 for the QR decomposition)
 - Each row of the vector index by $(i, x) \in \{1, \dots, n\} \times \mathcal{X}$
 - The functions f_i are “stacked” on top of one another
- “ $\infty \times n$ ” quasimatrix.¹ (used in Sections 2.2.3, 2.2.4, and 3.3.2 for the Maxvol algorithm).
 - Each row of the quasimatrix index by $x \in \mathcal{X}$
 - Each column of the quasimatrix indexed by $i \in \{1, \dots, n\}$
 - The functions f_i are placed “next” to one another
- “ $\infty \times \infty$ ” matrix for some separation F^k with $1 \leq k < d$ (used in Section 3.1 to study the maxvol property).
 - Rows indexed by tuples $(i, x_{\leq k}) \in \{1, \dots, n\} \times \mathcal{X}_{\leq k}$
 - Columns indexed by $x_{>k} \in \mathcal{X}_{>k}$

A collection of m vector-valued functions forms a matrix-valued function. Suppose we have m vector-valued functions $F_j : \mathcal{X} \rightarrow \mathbb{R}^n$ then a matrix-valued function $\mathcal{F} : \mathcal{X} \rightarrow \mathbb{R}^{n \times m}$ with m columns and n rows can be represented as

$$\mathcal{F} = [F_1 \ F_2 \ \dots \ F_m] \Leftrightarrow \mathcal{F}[:, j](x) = F_j(x)$$

¹Called a quasimatrix because it corresponds to a matrix of infinite rows and n columns. See e.g., [4, 32]

where column j of the matrix-valued function $\mathcal{F}[:, j]$ being equivalent to the vector-valued function F_j . Furthermore we view \mathcal{F} as analogous to a matrix of infinite rows and m columns. Each “row” of this matrix can be thought of the (index,value) pair (i, x) where $i \in \{1, \dots, n\}$ and $x \in \mathcal{X}$.

We now define the inner products between vector-valued and matrix-valued functions because they will be critical for defining the LU, QR, and SVD decompositions of matrix-valued functions. First, we recall that the inner product between two functions f and g is

$$\langle f, g \rangle = \int_{\mathcal{X}} f(x)g(x)dx$$

The inner product between two vector-valued functions $F : \mathcal{X} \rightarrow \mathbb{R}^n$ and $G : \mathcal{X} \rightarrow \mathbb{R}^n$ us defined as

$$(2.4) \quad \langle F, G \rangle = \sum_{i=1}^n \langle F[i](x), G[i](x) \rangle$$

We can use a similar definition for the inner product between two matrix-valued functions $\mathcal{F} : \mathcal{X} \rightarrow \mathbb{R}^{m \times n}$ and $\mathcal{G} : \mathcal{X} \rightarrow \mathbb{R}^{m \times n}$ as

$$(2.5) \quad \langle \mathcal{F}, \mathcal{G} \rangle = \sum_{i=1}^m \sum_{j=1}^n \langle \mathcal{F}[i, j](x), \mathcal{G}[i, j](x) \rangle,$$

where this inner product can be interpreted as the inner product between two flattened vector-valued functions, and it is analogous to the square of the Frobenius norm of a matrix. It is straightforward to verify that these definitions for inner products satisfy the necessary symmetry, linearity, and positive-definiteness conditions.

2.2.2. Multiplication. In addition to inner products, we can define products between arrays of functions (vector- and matrix-valued functions) and arrays of scalars (vectors and matrices) as shown in Table 1.

	$F : \mathcal{X} \rightarrow \mathbb{R}^n$	$\mathcal{F} : \mathcal{X} \rightarrow \mathbb{R}^{m \times n}$
$\mathbf{x} \in \mathbb{R}^n$	$g = F\mathbf{x} \implies$ $g(x) = \sum_{i=1}^n F[i](x)\mathbf{x}[i]$	$G = \mathcal{F}\mathbf{x} \implies$ $G[i](x) = \mathcal{F}[i, :](x)\mathbf{x}$
$\mathbf{A} \in \mathbb{R}^{n \times l}$	$G = F\mathbf{A} \implies$ $G[i](x) = F\mathbf{A}[:, i]$	$\mathcal{G} = \mathcal{F}\mathbf{A} \implies$ $\mathcal{G}[i, j](x) = \mathcal{F}[i, :](x)\mathbf{A}[:, j]$

TABLE 1

Definition of vector-valued function – vector (top left), vector-valued function – matrix products (bottom left), matrix-valued function – vector (top right), and matrix-valued function – matrix (bottom right) products.

Furthermore, we can define products for elements with different input spaces. Assume $\mathcal{X} \subset \mathbb{R}^d$ and $\mathcal{Y} \subset \mathbb{R}^d$, then products between vector- and matrix-valued functions result in functions on the space $\mathcal{X} \times \mathcal{Y}$. A summary of the operations and their notation is provided in Table 2.

We now turn to factorizations, or decompositions, of vector- and matrix-valued functions.

2.2.3. LU decomposition. For the cross-approximation and maxvol algorithms in the functional setting, we will require an LU decomposition of vector- and matrix-valued functions. The LU decomposition of a vector-valued function consists of $L = [\ell_1 \ell_2 \dots \ell_n]$, a vector-valued function that is “psychologically” lower triangular [32]; a matrix $\mathbf{U} \in \mathbb{R}^{r \times r}$, an upper triangular matrix; and a set of pivots $\mathbf{x} = [x_1 \ x_2 \ \dots \ x_n]$. A lower triangular vector-valued function is defined such that column k has zeros at all x_i for $i = 1 \dots k - 1$. Furthermore if $\ell_i(x_i) = 1$ then L is *unit lower triangular*.

	$F : \mathcal{X} \rightarrow \mathbb{R}^n$	$\mathcal{F} : \mathcal{X} \rightarrow \mathbb{R}^{m \times n}$
$H : \mathcal{Y} \rightarrow \mathbb{R}^n$	$g = FH \implies$ $g(x, y) =$ $\sum_{i=1}^n F[i](x)H[i](y)$	$G = \mathcal{F}H \implies$ $G[i](x, y) =$ $\sum_{j=1}^n \mathcal{F}[i, j](x)H[j](y)$
$\mathcal{H} \in \mathcal{Y} \rightarrow \mathbb{R}^{n \times k}$	$G = F\mathcal{H} \implies$ $G[i](x, y) =$ $\sum_{j=1}^n F[j](x)\mathcal{H}[j, i](y)$	$\mathcal{G} = \mathcal{F}\mathcal{H} \implies$ $\mathcal{G}(x, y) = \mathcal{F}(x)\mathcal{H}(y)$

TABLE 2

Definition of vector/vector-valued function product (top left), vector/matrix-valued function product (bottom left), matrix/vector-valued function product (top right), and matrix/matrix-valued function product (bottom right).

Finally, if $|\ell_k(x)| \leq |\ell_k(x_k)|$ for all $x \in \mathcal{X}$ then L is *diagonally maximal*. Using these definition we recall the definition of an LU decomposition of a vector-valued function according to [4, 32]

DEFINITION 2.1 (LU decomposition of a vector-valued function (Battles 2004)). *Let F be an $\mathcal{X} \rightarrow \mathbb{R}^n$ vector-valued function. An LU factorization of F is a decomposition with the form $F = LU$ where \mathbf{U} is upper triangular, and L is unit lower triangular and diagonally maximal.*

The LU decomposition may be computed using Gaussian elimination with row pivoting according to the algorithm in [32].

We can extend the definition of an LU decomposition to matrix valued functions. In particular, the decomposition will result in a “psychologically” lower triangular matrix-valued function \mathcal{L} ; an upper triangular matrix \mathbf{U} ; and a set of pivots $\alpha = [(i_1, x_1), (i_2, x_2) \dots (i_n, x_n)]$, where i_l denotes a particular row of \mathcal{L} . If we think of each column of \mathcal{L} as stacked functions, we see that each element of α is a location at which we will enforce the matrix-valued function to be zero. If we consider column k of \mathcal{L} , we obtain a vector-valued function $\mathcal{L}[:, k]$. A lower-triangular matrix-valued function is defined such that the column k has enforced zeros at all $[(i_1, x_1), (i_2, x_2), \dots (i_{k-1}, x_{k-1})]$ — e.g., $\mathcal{L}[:, 1]$ has no enforced zeros, $\mathcal{L}[:, 2]$ has an enforced zero at row i_1 and at the value of x_1 , etc. Furthermore, if $\mathcal{L}[i_k, k](x_k) = 1$ then \mathcal{L} is denoted as *unit lower triangular*, and if $|\mathcal{L}[i, k](x)| \leq |\mathcal{L}[i_k, k](x_k)|$ for all $x \in \mathcal{X}$ and for all $i \in [1 \dots n]$ then \mathcal{L} is diagonally maximal. Using these notions we define an LU decomposition of a matrix-valued function

DEFINITION 2.2 (LU decomposition of a matrix-valued function). *Let \mathcal{F} be an $\mathcal{X} \rightarrow \mathbb{R}^{m \times n}$ matrix-valued function. An LU factorization of \mathcal{F} is a decomposition with the form $\mathcal{F} = \mathcal{L}\mathbf{U}$ is a factorization where \mathbf{U} is upper triangular, and \mathcal{L} is unit lower triangular and diagonally maximal.*

We also implement the LU decomposition of a matrix-valued function using Gaussian elimination with row-pivoting.

2.2.4. QR decomposition. Another algorithm that will be necessary for function-train rounding and for cross-approximation of multidimensional functions is the QR decomposition of a quasi-matrix.

DEFINITION 2.3 (QR decomposition of a vector-valued function (Battles 2004)). *Let $F : \mathcal{X} \rightarrow \mathbb{R}^n$ be a vector-valued function. A QR factorization of F is a decomposition with the form $F = QR$ is one where the vector-value function Q consists of n orthonormal functions and \mathbf{R} is an upper triangular matrix with size $n \times n$. The computation of this QR factorization can be computed in a stable manner using a continuous extension of Householder triangularization [33]. In this paper, we also use the QR decomposition of a matrix-valued function.*

DEFINITION 2.4 (QR decomposition of a matrix-valued function). *Let $\mathcal{F} : \mathcal{X} \rightarrow \mathbb{R}^{m \times n}$ matrix-valued function. A QR factorization of \mathcal{F} is a decomposition with the form $\mathcal{F} = \mathcal{Q}\mathbf{R}$ is one where the matrix-valued function columns of \mathcal{Q} are orthonormal quasimatrices and \mathbf{R} is an upper triangular matrix $n \times n$.*

Since we have defined the inner product between vector-valued functions (2.4) we are able to also compute this factorization in a stable manner using Householder triangularization. We can consider

the rank of both vector-valued and matrix-valued functions as the number of non-zero elements of the diagonal of \mathbf{R} .

2.2.5. Singular value decomposition. For our theoretical results we will need to refer to the functional SVD. Let $\mathcal{Y} \times \mathcal{Z} \subset \mathbb{R}^d$.

DEFINITION 2.5 (Functional SVD). *Let $g : \mathcal{Y} \times \mathcal{Z} \rightarrow \mathbb{R}$. A singular value decomposition of g is*

$$(2.6) \quad g(y, z) = \sum_{j=1}^{\infty} \sigma_j u_j(y) v_j(z), \quad y \in \mathcal{Y} \quad z \in \mathcal{Z}$$

such that u_j are orthonormal scalar-valued functions, and can be arranged into a vector-valued function $U : \mathcal{X} \rightarrow \mathbb{R}^\infty$; $\sigma_1 \geq \sigma_2 \geq \dots \geq 0$; and v_j are orthonormal scalar-valued functions and can be arranged into a vector-valued function $V : \mathcal{Y} \rightarrow \mathbb{R}^\infty$.

In practice, the rank is truncated to some finite value r and in this case we have $U : \mathcal{Y} \rightarrow \mathbb{R}^r$, $\mathbf{S} \in \mathbb{R}^{r \times r}$ and $V : \mathcal{Z} \rightarrow \mathbb{R}^r$. In [32] the authors note that the concept of the SVD of a function has existed for a while [27, 28]. Recently [32] shows that if $g : [a, b] \times [c, d] \rightarrow \mathbb{R}$ is Lipschitz continuous, then the series in (2.6) converges absolutely and uniformly. In general, convergence can be assumed to be in L^2 .

It will be useful to use SVD to analyze certain separated views f^k of the multidimensional functions f . For example, we will see that the ranks of our function-train representation will be bounded by the ranks of the functional SVD of the separated functions $f^k : \mathcal{Y} \times \mathcal{Z} \rightarrow \mathbb{R}$, where we equate $\mathcal{Y} = \mathcal{X}_{\leq x}$ and $\mathcal{Z} = \mathcal{X}_{> x}$ in the above definition.

We now present a decomposition which is similar to the SVD for a vector-valued function. We will call the decomposition an *extended SVD* of a vector-valued function because it shares some properties with the functional SVD such as a separation rank and splits the vector-valued function into the sum of the products of one dimensional orthonormal elements (in particular a matrix-valued function and a vector-valued function). This decomposition appears in the proofs of Theorems 4.1 and 4.2 as well as in the skeleton decomposition of a multidimensional function described in Section 4.2.1

DEFINITION 2.6 (Extended SVD). *Let $G : \mathcal{Y} \times \mathcal{Z} \rightarrow \mathbb{R}^n$ be a vector-valued function consisting of a size n array of two-dimensional functions. An rank r extended SVD of $G(y, z)$ is a factorization*

$$(2.7) \quad G[i](y, z) = \sum_{j=1}^r \sigma_j U_j[i](y) v_j(z),$$

such that $U_j : \mathcal{Y} \rightarrow \mathbb{R}^n$ are orthonormal vector-valued functions that can be grouped into a matrix-valued function $\mathcal{U} : \mathcal{Y} \rightarrow \mathbb{R}^{n \times r}$ and $v_j : \mathcal{Z} \rightarrow \mathbb{R}$ are orthonormal scalar-valued functions that can be grouped into a vector-valued function $V : \mathcal{Z} \rightarrow \mathbb{R}^r$.

The main interpretation of this decomposition is that G contains n functions which have the same right singular vectors and different left singular vectors. We will exploit two properties of this decomposition for the proof of 3.6. The first is the fact that for any fixed \hat{z} the vector-valued function $\hat{G}[\hat{z}](y) \equiv G[\hat{z}](y, \hat{z})$ can be represented as a linear combination of the columns of $\mathcal{U}(y)$. Second, for any fixed \hat{y} and column \hat{i} the function $\hat{g}(\hat{z}) \equiv G[\hat{i}](\hat{y}, \hat{z})$ can be represented as a linear combination of the columns of V .

Again, the interpretation in the multidimensional case follows from the separated view of the vector-valued function. In particular if $X \subset \mathbb{R}^d$ then we can consider the extended SVD of a vector-valued function through its separated interpretation $F^k : \mathcal{Y} \times \mathcal{Z} \rightarrow \mathbb{R}$, where we equate $G = F^k$, $\mathcal{Y} = \mathcal{X}_{\leq k}$ and $\mathcal{Z} = \mathcal{X}_{> k}$ in the above definition.

3. Continuous skeleton/CUR decomposition. In Section 4.1.1 we will see that one can compute the FT representation of a function f through a sequence of singular value decompositions of various separated views of f . However, such an algorithm would require a prohibitive amount

of function evaluations. In this section we provide an alternative low-rank decomposition to the SVD. This alternative is a continuous version of the skeleton/CUR decomposition of a matrix and is critical for enabling the practical black-box approximation algorithm described in Section 4.2.1. In particular, we provide this continuous extension for vector-valued functions. The section is split into three parts. First we prove the existence of the continuous CUR decomposition. Then we motivate a construction of the decomposition based on the maximum volume concept. Finally, we describe a cross-approximation algorithm for computing the decomposition.

Before introducing the CUR decomposition we first introduce its components. In particular we need to formulate the concept of a vector-valued function fiber, an analogous concept to a row or column of a matrix. Let us consider a k -separated representation of a vector-valued function $F^k : \mathcal{X}_{\leq k} \times \mathcal{X}_{>k} \rightarrow \mathbb{R}^n$.

DEFINITION 3.1 (Column-fiber). *A column-fiber of a vector-valued function F^k is defined to be a vector-valued function $C_y : \mathcal{X}_{\leq k} \rightarrow \mathbb{R}^n$ obtained by fixing an element $x_{>k} = y$ — i.e., we have*

$$C_y[i](x_{\leq k}) = F^k[i](x_{\leq k}, y)$$

We can group a set of ℓ column-fibers together to obtain a matrix-valued function $\mathcal{C}_{\mathbf{y}}$.

DEFINITION 3.2 (Set of column-fibers). *A set of column-fibers of a vector-valued function F^k is defined to be the matrix-valued function $\mathcal{C}_{\mathbf{y}} : \mathcal{X}_{\leq k} \rightarrow \mathbb{R}^{n \times \ell}$ obtained by choosing a vector of ℓ values from $\mathcal{X}_{>k}$ termed $\mathbf{y} = [y_1, \dots, y_\ell]$ where $y_i \in \mathcal{X}_{>k}$, i.e. —*

$$\mathcal{C}_{\mathbf{y}} = [C_{y_1} \ C_{y_2} \ \dots \ C_{y_\ell}]$$

We have corresponding definitions for row-fibers and a set of row-fibers.

DEFINITION 3.3 (Row-fiber). *A row-fiber of a vector-valued function F^k is defined to be a scalar-valued function $r_\alpha : \mathcal{X}_{>k} \rightarrow \mathbb{R}$ obtained by fixing an (index,value) pair $\alpha = (i, x_{\leq k}) \in \{1, \dots, n\} \times \mathcal{X}_{\leq k}$ to obtain*

$$r_\alpha(x_{>k}) = F^k[i](x_{\leq k}, x_{>k})$$

DEFINITION 3.4 (Set of row-fibers). *A set of row-fibers of a vector-valued function F^k is obtained by choosing a vector of ℓ tuples $\boldsymbol{\alpha} = [(i_1, x_1), \dots, (i_\ell, x_\ell)]$ where $(i_j, x_j) \in \{1, \dots, n\} \times \mathcal{X}_{\leq k}$ to obtain*

$$R_{\boldsymbol{\alpha}}(x_{>k}) = [r_{\boldsymbol{\alpha}[1]} \ r_{\boldsymbol{\alpha}[2]} \ \dots \ r_{\boldsymbol{\alpha}[\ell]}]$$

We are now ready to define the existence of a skeleton/CUR decomposition.

3.1. Existence of the skeleton decomposition of vector-valued functions. Here, we now define and prove the existence of a skeleton decomposition of a vector-valued function that has a finite extended SVD rank when viewed as a separated vector-valued function. The existence of a skeleton decomposition for scalar-valued functions can be obtained by choosing $n = 1$. In fact, the skeleton decomposition of a function has already been used, for example, in [5, 6]. These previous results study the properties of the CUR decomposition through smoothness assumptions on the function. Our results, in contrast, will discuss the CUR in the context of the low-rank properties of the function.

DEFINITION 3.5 (Skeleton/CUR decomposition of a vector-valued function). *Let $F : \mathcal{X} \rightarrow \mathbb{R}^n$ be a vector-valued function. The rank ℓ skeleton decomposition of separation F^k is a factorization of the type*

$$(3.1) \quad F^k[i](x_{\leq k}, x_{>k}) = \mathcal{C}_{\mathbf{y}}[i, :](x_{\leq k}) \mathbf{G} R_{\boldsymbol{\alpha}}(x_{>k})$$

where $\mathcal{C}_{\mathbf{y}} : \mathcal{X} \rightarrow \mathbb{R}^{n \times \ell}$ is a matrix-valued function representing a set of ℓ column-fibers (Definition 3.2), \mathbf{G} is a $\ell \times \ell$ matrix, and $R_{\alpha} : \mathcal{Y} \rightarrow \mathbb{R}^{\ell}$ is a vector-valued function representing a set of ℓ row-fibers (Definition 3.4).

THEOREM 3.6. *Let $F : \mathcal{X} \rightarrow \mathbb{R}^n$ be a vector-valued functions and let $F^k : \mathcal{X}_{\leq k} \times \mathcal{X}_{>k} \rightarrow \mathbb{R}^n$ be its k -separated form for any $1 \leq k < d$. If F^k has a rank r extended SVD decomposition, then a rank r skeleton decomposition (3.1) of F exists.*

Proof. Choose a vector of row-indices α (see Definition 3.4) such that the vector-valued function R_{α} contains at least r linearly independent scalar-valued functions $r_{\alpha[i]}$. Next choose a vector of column indices \mathbf{y} (see Definition 3.2) such that the matrix-valued function $\mathcal{C}_{\mathbf{y}}$ contains at least r linearly independent vector-valued functions C_{y_i} .

Then define the matrix \mathbf{F} formed through the intersection of the row and column fibers. Elementwise, we have

$$(3.2) \quad \mathbf{F}[l, j] = \mathcal{C}_{\mathbf{y}}[i_l, j](x_l) = R_{\alpha}[l](y_j) \quad (i_l, x_l) = \alpha[l]$$

and we obtain a skeleton decomposition

$$\tilde{F}(x_{\leq k}, x_{>k}) = \mathcal{C}_{\mathbf{y}}(x_{\leq k}) \mathbf{F}^{\dagger} R_{\alpha}(x_{>k})$$

We now seek to show that \tilde{F} and F^k are pointwise equal.

We show $F^k = \tilde{F}$ by decomposing the space $((\{1, \dots, n\} \times \mathcal{X}_{\leq k}) \times \mathcal{X}_{>k})$ into four partitions: $(\alpha \times \mathbf{y})$, $(\sim \alpha \times \mathbf{y})$, $(\alpha \times \sim \mathbf{y})$, and $(\sim \alpha \times \sim \mathbf{y})$, where $\sim \alpha = (\{1, \dots, n\} \times \mathcal{X}_{\leq k}) \setminus \alpha$ and $\sim \mathbf{y} = \mathcal{X}_{>k} \setminus \mathbf{y}$. We will also rely on the following property of the Moore-Penrose pseudoinverse \mathbf{A}^{\dagger} of a matrix \mathbf{A} :

$$(3.3) \quad \mathbf{A} \mathbf{A}^{\dagger} \mathbf{A} = \mathbf{A}.$$

We now consider the first partition $(\alpha \times \mathbf{y})$, where we see

$$\begin{aligned} \tilde{F}[\mathbf{i}](\mathbf{x}, \mathbf{y}) &= \mathcal{C}_{\mathbf{y}}[\mathbf{i}, :](\mathbf{x}) \mathbf{F}^{\dagger} R_{\alpha}(\mathbf{y}) \\ &= F^k[\mathbf{i}](\mathbf{x}, \mathbf{y}), \end{aligned}$$

where the first equality follows from the definition of \tilde{F} and the second equality follows from (3.3).

We move on to the space $\sim \alpha \times \mathbf{y}$. By definition of linear independence and the SVD rank of a function, $\forall (i, x) \in -(\mathbf{i}, \mathbf{x}) \exists \mathbf{q}_{i,x} : \{1, \dots, n\} \times \mathcal{X}_{\leq k} \rightarrow \mathbb{R}^{\ell}$ s.t.

$$(3.4) \quad F^k[\mathbf{i}](x, \hat{\mathbf{y}}) = \mathbf{q}_{i,x} R_{\alpha}(\hat{\mathbf{y}}) \quad \forall \hat{\mathbf{y}} \subset \mathcal{X}_{>k}$$

In other words, every quasimatrix $F(x, \hat{\mathbf{y}})$ can be written as a linear combination of the functions of the matrix-valued function $R_{\alpha}(\hat{\mathbf{y}})$. This property follows directly from the fact that $R_{\alpha}(\mathbf{y})$ consists of at least r linearly independent functions and that F^k has extended SVD rank r .

Using (3.4) we can now show that (3.1) holds for $(\hat{\alpha}, \mathbf{y}) \in (\sim \alpha \times \mathbf{y})$,

$$\begin{aligned} \tilde{F}[\mathbf{i}](x, \mathbf{y}) &= \mathcal{C}_{\mathbf{y}}[\mathbf{i}, :](x) \mathbf{F}^{\dagger} R_{\alpha}(\mathbf{y}) \\ &= \mathbf{q}_{i,x} R_{\alpha}(\mathbf{y}) \mathbf{F}^{\dagger} R_{\alpha}(\mathbf{y}) \\ &= \mathbf{q}_{i,x} \mathbf{F} \mathbf{F}^{\dagger} \mathbf{F} \\ &= \mathbf{q}_{i,x} \mathbf{F} \\ &= F^k[\mathbf{i}](x, \mathbf{y}) \end{aligned}$$

where the first equality comes from the definition of \tilde{F} , the second comes from (3.4) and using the fact that $F^k[\mathbf{i}](x, \mathbf{y}) = \mathcal{C}_{\mathbf{y}}[\mathbf{i}, :](x)$, the third comes from the definition of \mathbf{F} , using (3.4), the fourth equality comes from using (3.3), and the fifth also comes from (3.4).

We can proceed with a symmetric argument for $(\alpha, y) \in (\alpha \times \sim y)$. Linear independence requires $\forall y \in \sim y \exists \tilde{q}_y : \mathcal{X}_{>k} \rightarrow \mathbb{R}^\ell$ s.t.

$$(3.5) \quad F^k[\hat{i}](\hat{x}, y) = C_y[\hat{i}, :](\hat{x})\tilde{q}_y \quad \forall (\hat{i}, \hat{x}) \subset (\{1, \dots, n\} \times \mathcal{X}_{\leq k})$$

Using (3.5) a symmetrical argument equivalence for elements of the desired space $(\alpha \times \sim y)$.

Finally, we show (3.1) holds pointwise for all points $(x, y) \in (\sim \alpha \times \sim y)$. First, by (3.4) we have $F^k[i](x, \hat{y}) = q_{i,x}R_\alpha(\hat{y})$. Next, we can obtain

$$(3.6) \quad F^k[i](x, y) = q_{i,x}R_\alpha(\hat{y}) = q_{i,x}C_y[i, :](x)\tilde{q}_y = q_{i,x}\mathbf{F}q_y,$$

where the second equality follows from (3.5) and the third equality follows from the definition of \mathbf{F}

Now we show that the application of (3.4) and (3.5) to (3.1) yields the desired result

$$\begin{aligned} \tilde{F}[i](x, y) &= C_y[i, :](x)\mathbf{F}^\dagger R_\alpha(y) \\ &= q_{i,x}R_\alpha(y)\mathbf{F}^\dagger C_y[i, :](x)\tilde{q}_y \\ &= q_{i,x}\mathbf{F}\tilde{q}_y \\ &= F^k[i](x, y), \end{aligned}$$

where the first equality follows from the definition of \tilde{F} the second follows from an application of (3.4) and (3.5), the third follows from the Moore-Penrose pseudoinverse (3.3), and the final equality follows from (3.6).

□

The proof is constructive and requires one to choose $\ell > r$ linearly independent column- and row-fibers. Furthermore, if one defines a matrix \mathbf{F} that lies at the intersection of these values of i, x, y and contains elements $\mathbf{F}[a, b] = F[i_a](x_a, y_b)$. Then one obtains a valid skeleton decomposition of F with $\mathbf{G} = \mathbf{F}^\dagger$.

A natural question to ask is what happens when one wants to obtain an approximation with $\ell < r$ for a particular separated representation F^k of F . In particular, how do we choose y and α so that we obtain a well behaved approximation. The answer is to choose them so that matrix \mathbf{F} lying at the intersection of the rows and column fibers has maximum volume amongst all possible combinations of ℓ rows and fibers.

3.2. Optimality of Maxvol. Recall that we are now considering the case of $\ell < r$, or the case where we would like to use fewer terms in the approximation than the actual rank of the function. This situation arises often because many functions can be considered to be *approximately* low-rank. Here an approximately low-rank function is one that may have an infinite number of singular values, but that the singular values quickly decay to zero – e.g.,

$$(3.7) \quad f(x, y) = \sum_{i=1}^{\ell} \sigma_i u_i(x) v_i(y) + g(x, y), \quad \|g(x, y)\|_{L_2} = \epsilon$$

Such a function can potentially be accurately represented with a finite rank approximation. In particular, suppose we have chosen a separation k , an approximation rank $\ell < r$ and a CUR decomposition defined by a set of column fibers y and a set of row fibers α to form an approximation

$$(3.8) \quad \tilde{F}^k[i](x_{\leq k}, x_{>k}) = C_y[i, :](x_{\leq k})\mathbf{G}R_\alpha(x_{>k}).$$

The goal of this sections is to bound $\|F - \tilde{F}\|$ and to show that this bound holds when the matrix \mathbf{G}^\dagger in the skeleton decomposition is chosen to be a maximum-volume sub-matrix of F , where the volume of a matrix refers to the modulus of its determinant. A sub-matrix of a vector-valued function is defined as

DEFINITION 3.7 (Sub-matrix of a vector-valued function). *Let $F^k : \mathcal{X}_{\leq k} \times \mathcal{X}_{>k} \rightarrow \mathbb{R}^n$ be a k -separated view of the vector-valued function F . Define a set of ℓ indices $\alpha = [(i_1, x_1), (i_2, x_2), \dots, (i_\ell, x_\ell)]$*

such that $(i_a, x_a) \in \{1, \dots, \ell\} \times \mathcal{X}_{\leq k}$ and $(x_a, i_a) \neq (x_b, i_b)$ for $a \neq b$. Similarly define a set of y indices $\mathbf{y} = [y_1, y_2, \dots, y_\ell]$ such that $y_a \in \mathcal{X}_{> k}$. A square sub-matrix of F formed at the intersection of these indices is given as $\mathbf{F} \in \mathbb{R}^{\ell \times \ell}$ with elements

$$(3.9) \quad \mathbf{F}[a, b] = F[i_a](x_a, y_b) \quad \text{for} \quad 1 \leq a, b \leq \ell$$

An intuition for this definition lies from viewing F as an “ $\infty \times \infty$ ” matrix with rows indexed by $(i, x_{\leq k})$ and columns indexed by $x_{> k}$. Using this definition we can define a maximum volume sub-matrix.

DEFINITION 3.8 (Maximum volume sub-matrix). *A sub-matrix \mathbf{F} of a vector-valued function F with a specified k -separation F^k is defined to have maximum volume if it has maximum modules of determinant among all possible sub-matrices of F^k . The maximum volume sub-matrix is denoted as \mathbf{F}_s .*

The following theorem is proven for matrices in [13] and we extend it to vector-valued functions of two dimensions.

THEOREM 3.9. *Let $F : \mathcal{X} \rightarrow \mathbb{R}^n$ be a continuous vector-valued function and F^k be a separated form that has an extended SVD decomposition with uniformly bounded singular functions $\max_{i,j,x_{\leq k}} |U_j[i](x_{\leq k})| < a$ and $\max_{j,x_{> k}} |v_j(x_{> k})| < a$ (recall Equation (2.6)). Furthermore, suppose that \tilde{F} is a skeleton approximation of F^k and that $\mathbf{F}_s \in \mathbb{R}^{\ell \times \ell}$ is a non-singular maximum volume sub-matrix of F formed by the intersection of ℓ row and column fibers of that form \tilde{F} . Then it holds that*

$$(3.10) \quad \|F - \tilde{F}\|_C \leq Ma^2(k+1)^{1+2\varepsilon} \sigma_{k+1}$$

where M is a constant not depending on k and $\varepsilon > \frac{1}{2}$ are defined in Appendix A. $\|\cdot\|_C$ denotes that maximum in modulus element – i.e., $\|F\|_C = \max_{i,x_{\leq k},x_{> k}} F[i](x_{\leq k}, x_{> k})$.

Proof. Let us denote $E \equiv F - \mathcal{C}\mathbf{F}_s^\dagger R$. Now we bound the absolute value of each element of E . Consider any $k+1 \times k+1$ sub-matrix of F formed by augmenting \mathbf{F}_s by one row and column of elements from F

$$(3.11) \quad \mathbf{F} = \begin{bmatrix} \mathbf{F}_s & \mathbf{b} \\ \mathbf{c}^T & a \end{bmatrix},$$

where for each possible y value we have a vector \mathbf{b} such that each of its elements can be written as $\mathbf{b}[j] = F[i_j](x_j, y)$ for $j = 1 \dots k$. Relatedly, for each combination of (i, x) we have the vector \mathbf{c} with elements $\mathbf{c}[j] = F[i](x, y_j)$. The values for a similarly arise from all possible combinations of (i, x, y) .

Now define $\gamma = a - \mathbf{c}^T \mathbf{F}_s^{-1} \mathbf{b}$ and note that γ corresponds to an element of E obtained by the combination (i, x, y) chosen to augment \mathbf{F}_s . Also, $\gamma = 0$ iff \mathbf{F} is singular due to the following property of the determinant,

$$(3.12) \quad \det \left(\begin{bmatrix} \mathbf{F}_s & \mathbf{b} \\ \mathbf{c}^T & d \end{bmatrix} \right) = \det(\mathbf{F}_s) \det(d - \mathbf{c}^T \mathbf{F}_s^{-1} \mathbf{b})$$

for any vectors \mathbf{b}, \mathbf{c} and scalar d as long as \mathbf{F}_s is invertible. If $\gamma = 0$, then the element of E to which it refers is zero.

Now we consider the case where $\gamma \neq 0$ and that \mathbf{F} is non-singular. Using property (3.12), we see

$$|\det(\mathbf{F})| = |\det(\mathbf{F}_s)| |\gamma|$$

Furthermore, recall that we can write the inverse of a matrix using the matrix of cofactors \mathbf{C}

$$(3.13) \quad \mathbf{F}^{-1} = \frac{1}{\det(\mathbf{F})} \mathbf{C}^T$$

The maximal volume property of \mathbf{F}_s now implies that $\|\mathbf{C}^T\|_C = |\det(\mathbf{F}_s)|$. Together with Equation (3.13) this implies

$$(3.14) \quad \|\mathbf{F}^{-1}\|_C = \frac{1}{|\det(\mathbf{F})|} |\det(\mathbf{F}_s)| = |\gamma^{-1}|.$$

Now we can use the Frobenius norm on \mathbf{F} to obtain a bound using its singular values

$$\|\mathbf{F}^{-1}\|_F = \sqrt{\sum_{i=1}^{k+1} \sigma_i^{-2}(\mathbf{F})} \leq \sqrt{\|\mathbf{F}^{-1}\|_C^2 (k+1)^2}$$

which implies

$$\begin{aligned} \sigma_{k+1}^{-2}(\mathbf{F}) &\leq \sum_{i=1}^{k+1} \sigma_i^{-2}(\mathbf{F}) \leq \|\mathbf{F}^{-1}\|_C^2 (k+1)^2 \\ \sigma_{k+1}(\mathbf{F})(k+1) &\geq \|\mathbf{F}^{-1}\|_C^{-1} \end{aligned}$$

Now using (3.14) we have $\sigma_{k+1}(\mathbf{F})(k+1) \geq |\gamma|$. From Lemma A.1 in Appendix A we have the inequality

$$\sigma_{k+1}(\mathbf{F}) \leq Ma^2(k+1)^{2\varepsilon} \sigma_{k+1}(F),$$

for $\varepsilon > 1/2$. This inequality provides us with the final result.

□

This theorem implies that utilizing a maximum volume sub-matrix \mathbf{F}_s allows us to bound the error of the skeleton decomposition by a constant factor greater than the error obtained by the SVD, the optimal low rank decomposition. This result is different from the result of [5] in which it was proved that the error between a function and its skeleton decomposition can be bounded by a factor of the maximal value of an interpolating Lagrange polynomial. In particular, this result refocuses the problem on *rank* instead of smoothness.

We now discuss an algorithm for computing the skeleton decomposition by seeking a maximum volume cross matrix.

3.3. Cross-approximation and Maxvol. We now describe an algorithm for computing the skeleton decomposition of a vector-valued function. Computing the skeleton decomposition of a matrix is the subject of much current research. Some approaches [10, 19, 8] work based on random sampling of the rows and columns of the matrix. Another group of methods attempt to explicitly find the rows and columns which maximize the volume of the cross matrix. In [14, 13] the authors describe how a skeleton decomposition with a maximum volume cross matrix is quasioptimal. We will follow this second route as it has been successfully extended to the tensor-train case in [22] and proved to be quasioptimal in [26]. We now focus on extending the maximum volume based algorithms to the continuous, functional, case.

3.3.1. Cross approximation of a vector-valued function. Cross approximation of a vector-valued function in skeleton decomposition form is a fairly straight forward row-column alternating algorithm. The pseudocode for the algorithm is given by Algorithm 1. The algorithm works by first fixing a set of column fibers \mathbf{y} as defined computing a set of rows α to maximize the volume of the matrix-valued function $\mathcal{C}_{\mathbf{y}}$. Next, the set of row-fibers α are fixed and a new set of column

fibers \mathbf{y} is found which maximizes the volume of a sub-matrix of R_α . For a definition of row and column fibers of a vector-valued function refer to Section 3. The algorithm continuous alternating between rows and columns until convergence. The algorithm concludes when the difference successive approximations falls below a particular tolerance.

Algorithm 1 requires several sub-routines: **qr-mvf** refers to the QR decomposition of a matrix-valued function and is computed using Householder triangularization; **qr-vvf** refers to a QR decomposition of a vector-valued function and is also computed using Householder triangularization; **maxvol-mvf** refers to the Algorithm 2 discussed in Section 3.3.2.

Algorithm 1 Cross-approximation of a vector-valued function of two dimensions

Require: Separated view $F^k : \mathcal{X}_{\leq k} \times \mathcal{X}_{> k} \rightarrow \mathbb{R}^r$ of a vector-valued function F ; Rank estimate r ;
Initial colum fibers $\mathbf{y} = [y_1, y_2, \dots, y_r]$; Stopping tolerance $\delta_{cross} > 0$
Ensure: (α, \mathbf{y}) such that sub-matrix \mathbf{F} has “large” volume

- 1: $\delta = \delta_{cross} + 1$
- 2: $F_{(0)} = 0$
- 3: $k = 1$
- 4: $\mathcal{C}(x_{\leq k}) = [F^k(x_{\leq k}, y_1) \ F^k(x_{\leq k}, y_2) \ \dots \ F^k(x_{\leq k}, y_r)]$
- 5: **while** $\delta \leq \delta_{cross}$ **do**
- 6: $Q\mathbf{T} = \text{qr-mvf}(\mathcal{C})$
- 7: $\alpha = \text{maxvol-mvf}(Q)$
- 8: $R(x_{> k}) = [F^k[i_1](x_1, x_{> k}) \ \dots \ F^k[i_r](x_r, x_{> k})]$ where $(i_j, x_j) = \alpha[j]$
- 9: $Q\mathbf{T} = \text{qr-vvf}(R)$
- 10: $\mathbf{y} = \text{maxvol-mvf}(Q)$ # Interpret Q as a $1 \times r$ matrix-valued function.
- 11: $\mathcal{C}(x) = [F(x, y_1) \ F(x, y_2) \ \dots \ F(x, y_r)]$
- 12: $\hat{Q} = [Q(y_1) \ Q(y_2) \ \dots \ Q(y_r)]$
- 13: $F_{(k)}(x, y) = \mathcal{C}(x)\hat{Q}^\dagger Q(y)$
- 14: $\delta = ||F_{(k)} - F_{(k-1)}|| / ||F_{(k)}||$
- 15: $k = k + 1$
- 16: **end while**

We would like to point out the distinction between this row-column alternating algorithm and *adaptive* cross approximation algorithms such as [5]. Adaptive cross-approximation algorithms for approximation of two-dimensional functions rely on building up a set of x and y indices by sequentially choosing points which maximize the volume. In other words, they do not require one to prescribe a rank, nor do they require one to evaluate entire function fibers. These algorithms, which are equivalent to LU decomposition with complete pivoting, use two-dimensional optimization methods to seek these the optimal locations for function evaluation (which become the pivots). Such a methodology has difficulty extending to the multidimensional case because it would require us to optimize over d dimensions, a potentially expensive and difficult procedure.

3.3.2. Maxvol computation. Algorithm 2 provides the pseudocode for computing the maximum volume sub-matrix of a matrix-valued function, and it mirrors the algorithm provided in [12] for “tall and skinny” matrices. The algorithm attempts to find a sub-matrix which is *dominant* because a dominant sub-matrix has volume which is “close” to the maximum volume sub-matrix. In this section we always assume that $\text{rank}[\mathcal{R}] = r$ for a matrix-valued function $\mathcal{R} : \mathcal{X} \rightarrow \mathbb{R}^{n \times r}$, where the notion of rank of a matrix-valued function is defined in Section 2.2.4. Also, all of the results can be generalized to vector-valued functions by assuming that $n = 1$.

DEFINITION 3.10 (Sub-matrix of a vector-valued function). *Let $\mathcal{A} : \mathcal{X} \rightarrow \mathbb{R}^{n \times r}$ be a matrix-valued function, and define a set of r tuples $[(i_1, x_1), (i_2, x_2), \dots, (i_r, x_r)]$ such that $(i_k, x_k) \in \{1, \dots, n\} \times \mathcal{X}$, and $(i_k, x_k) \neq (i_l, x_l)$ for $k \neq l$ for all $k, l = 1 \dots r$. A rectangular sub-matrix $\mathbf{A} \in \mathbb{R}^{r \times r}$ of \mathcal{A} has elements $\mathbf{A}[k, j] = \mathcal{A}[i_k, j](x_k)$. Note the critical distinction between Definitions 3.7 and 3.10 is*

that Definition 3.7 defines a sub-matrix for a vector-valued function in a separated view F^k whereas Definition 3.10 refers to the maximal volume sub-matrix of a matrix-valued function. In particular, the relationship between these two definitions lies in the context of a set of column fibers, where Definition 3.10 refers to a matrix-valued function arising from a set of column fibers of a separated view F^k . Next, we need the notion of a dominant sub-matrix.

DEFINITION 3.11. *Let us define a matrix-valued function $\mathcal{R} : \mathcal{X} \rightarrow \mathbb{R}^{n \times r}$ such that $R = \mathcal{A}\mathbf{A}_s^{-1}$. The $r \times r$ sub-matrix \mathbf{A}_s is dominant if for all values $(i, x, k) \in \{1, \dots, n\} \times \mathcal{X} \times [1, \dots, r]$ we have $|R[i, k](x)| \leq 1$.*

LEMMA 3.12. *For a matrix-valued function $\mathcal{R} : \mathcal{X} \rightarrow \mathbb{R}^{n \times r}$ the maximum volume sub-matrix $\mathbf{A}_s \in \mathbb{R}^{r \times r}$ is dominant.*

Proof. The proof follows exactly the case for matrices given in [12].

Suppose, without loss of generality, that we rearrange the “rows” of \mathcal{R} such that we have

$$\mathcal{A}\mathbf{A}_s^{-1} = \begin{bmatrix} \mathbf{I}_{r \times r} \\ \mathcal{Z} \end{bmatrix} = \mathcal{B},$$

where \mathbf{A}_s is the maximum-volume sub-matrix of \mathcal{A} .

Now recall that $\det(\mathbf{C}) = \det(\mathbf{C})\det(\mathbf{D})$ for square matrices of equal size. This means that multiplying \mathcal{A} by a nonsingular $r \times r$ matrix B does not change the ratio of determinants of any pair of $r \times r$ sub-matrices in \mathcal{A} since they just get scaled by $\det(B)$. This fact implies that the upper sub-matrix $\mathbf{I}_{r \times r}$ is a maximum-volume sub-matrix in \mathcal{B} and it is dominant in \mathcal{B} iff \mathbf{A}_s is dominant in \mathcal{A} .

If \mathbf{A}_s was chosen such that $\mathcal{B}[i, j](x') > 1$ for some choice of (i, j, x) then we would be able to increase the volume of the upper sub-matrix by swapping the row $\mathcal{B}[i, :](x)$ with row j in the upper $r \times r$ sub-matrix of \mathcal{B} . The new upper sub-matrix of B will have $\det(B_{upper}) = \mathcal{B}[i, j](x)$ instead of $\det(\mathbf{I}_{r \times r}) = 1$. Thus we see that $\mathbf{I}_{r \times r}$ would not be the maximum volume sub-matrix in \mathcal{B} and thus would \mathbf{A}_s is not the maximum volume sub-matrix in \mathcal{A} . \square

This proof implies an algorithm which switches “rows” of \mathcal{A} (recall these are given by (index, x-value) combinations), until all the elements of \mathcal{B} are less than 1. This is exactly what the operations in lines 5 and 8 in Algorithm 2 are doing.

Furthermore, analogously to Lemma 2 in [12] we can show that any dominant sub-matrix cannot have volume that is too much smaller than the maximum volume sub-matrix.

LEMMA 3.13. *For any full rank matrix-valued function $\mathcal{A} : \mathcal{X} \rightarrow \mathbb{R}^{n \times r}$*

$$|\det(\mathbf{A}_d)| \geq \frac{|\det(\mathbf{A}_s)|}{r^{r/2}}$$

The proof is almost equivalent to the one in [12] and so is omitted for brevity.

Together, Lemmas 3.12 and 3.13 imply that searching for a maximum volume sub-matrix of a matrix-valued function \mathcal{A} through a row switching scheme described in Algorithm 2 is a good idea. The pivoted LU decomposition, resulting in pivots (i, \mathbf{x}) , is first performed to find linearly independent “rows” of \mathcal{A} .

4. Function-train decomposition. We now turn to the problem of deriving and computing the function-train (FT) decomposition. The existence of the FT has been tackled before in [7] in the context of spectral tensor-trains. However, the results here present the relation of the FT ranks to the ranks of the unfolding (or separated views) functions and are direct continuous analogues to the theory provided in [23, 22].

Recall that $\mathcal{X} = [a_1, b_1] \times \dots \times [a_d, b_d]$. A function-train is defined by a set of d - matrix-valued functions $\{\mathcal{F}_1(x_1), \mathcal{F}_2(x_2), \dots, \mathcal{F}_d(x_d)\}$ and a set of FT-ranks $\mathbf{r} = [r_0, r_1, \dots, r_d]$ such that $r_0 = r_d = 1$ and $\mathcal{F}_i : [a_i, b_i] \rightarrow \mathbb{R}^{r_{i-1} \times r_i}$ for $i = 1, \dots, d$. Furthermore, an evaluation of a function $f : \mathcal{X} \rightarrow \mathbb{R}$ in FT format $\hat{f} : \mathcal{X} \rightarrow \mathbb{R}$ is obtained through a sequence of vector-matrix products

$$(4.1) \quad \hat{f}(x) = \mathcal{F}_1(x_1)\mathcal{F}_2(x_2) \dots \mathcal{F}_d(x_d).$$

Algorithm 2 maxvol-mvf: Maximum volume of matrix-valued function

Require: $\mathcal{A} : \mathcal{X} \rightarrow \mathbb{R}^{n \times r}$, a matrix-valued function.

Ensure: (\mathbf{i}, \mathbf{x}) such that $\mathbf{A} = \mathcal{A}[\mathbf{i}, :](\mathbf{x})$ is a sub-matrix with with quasi-maximum volume

```

1:  $L, U, (\mathbf{i}, \mathbf{x}) = \text{lu-mvf}(A)$  # LU decomposition of a matrix-valued function
2:  $\delta = 2$ 
3: while  $\delta > 1$  do
4:    $\mathbf{A} = \mathcal{A}[\mathbf{i}, :](\mathbf{x})$ 
5:    $\mathbf{x}^*, \mathbf{i}^*, \mathbf{j}^* = \arg \max_{(x, i, j)} \mathcal{A}[\mathbf{i}, :](x) \mathbf{A}^\dagger[:, j]$ 
6:    $\delta = \mathcal{A}[\mathbf{i}, :](\mathbf{x}^*) \mathbf{Q}^\dagger[:, \mathbf{j}^*]$ 
7:   if  $\delta > 1$  then
8:      $\mathbf{x}[\mathbf{j}^*] = \mathbf{x}^*, \mathbf{i}[\mathbf{j}^*] = \mathbf{i}^*$ 
9:   end if
10: end while

```

It will also be helpful to think of these cores as a matrix of one dimensional functions — e.g.,

$$(4.2) \quad \hat{f}(x) = \begin{bmatrix} \hat{f}_1^{(1,1)} & \dots & \hat{f}_1^{(1,r_1)} \end{bmatrix} \begin{bmatrix} \hat{f}_2^{(1,1)} & \dots & \hat{f}_2^{(1,r_2)} \\ \vdots & & \vdots \\ \hat{f}_2^{(r_1,1)} & \dots & \hat{f}_2^{(r_1,r_2)} \end{bmatrix} \dots \begin{bmatrix} \hat{f}_d^{(1,1)} \\ \vdots \\ \hat{f}_d^{(r_{d-1},1)} \end{bmatrix},$$

(4.3)

where $\hat{f}_k^{(i,j)} : [a_k, b_k] \rightarrow \mathbb{R}$.

As we will prove in this section, the FT of a function f inherits many of the properties of the separated views f^k . The separated views f^k are equivalent to the notions of folding functions in the TT literature and we will use these terms interchangeably. For example, if the SVD of each of these f^k exists, then we can prove the FT exists. Also, these SVD ranks will upper bounds the FT rank r_k . In Section 4.1 we discuss the existence of the FT in the case where f^k have finite rank and are approximately low-rank, in Section 4.2.1 we discuss the computation of \hat{f} for fixed ranks \mathbf{r} through a higher dimensional cross-approximation algorithm, and in Section 4.2.2 we discuss a rank adaptation scheme with a continuous version of rounding [22].

4.1. Existence of the function-train decomposition. We now discuss the existence of the function-train for situations in which the k –separated views of a multidimensional functions are either finite-rank or approximately low-rank.

4.1.1. Finite-rank case. We start by proving that any function with finite rank unfoldings can be exactly represented by a FT with bounded ranks.

THEOREM 4.1. *If each separated view f^k of form (2.1) of a d -dimensional function f has a finite rank SVD(2.6) with ranks*

$$(4.4) \quad \text{rank } f^k = r_k$$

then there exists an FT decomposition \hat{f} of form (4.3) with FT-ranks not higher than r_k such that $\|\hat{f} - f\|_{L^2} = 0$.

Proof. We start with unfolding function f^k . Since its rank is r_1 , the function can be decomposed as

$$(4.5) \quad f^1(x_1, x_{>1}) = U_1(x_1) V_1(x_{>1}),$$

where $U_1 = [u_1^{(1)}, \dots, u_r^{(1)}]$ is a vector-valued function with orthogonal columns and $V_1 = [v_1^{(1)}, \dots, v_r^{(1)}]$ is a vector-valued function with orthonormal columns. This type of decomposition can arise from the functional SVD (2.6) in which the singular values have been absorbed into U_1

Using these orthogonality conditions we can define each $v_i^{(1)}$ as

$$(4.6) \quad v_i^{(1)}(x_{>1}) = \frac{\langle f^1(s, x_{>1}), u_i^{(1)}(s) \rangle}{\langle u_i^{(1)}(s), u_i^{(1)}(s) \rangle} = \langle f^1(s, x_{>1}), w_i(s) \rangle$$

We will now show that both $\text{rank} \left[(v_i^{(1)})^k \right] \leq r_k$ for $k = 2, \dots, d-1$ and that $(V_1)^k$ is also “low rank” according to the extended SVD definition (2.7).

We start by noting that unfolding k (or separated view f^k of f is

$$(4.7) \quad f^k(x_{\leq k}, x_{>k}) = F(x_{\leq k})G(x_{>k}),$$

where F and G are both vector-valued functions consisting of r_k functions. Now we plug these results into the definition of the k -th unfolding of $v_i^{(1)}$

$$\begin{aligned} (v_i^{(1)})^k &= v_i^{(1)}(\{x_2, \dots, x_k\}, \{x_{k+1}, \dots, x_d\}) \\ &= \langle f_1(s, x_{>1}), w_i(s) \rangle \\ &= \langle F(s, x_2, \dots, x_k)G(x_{>k}), w_i(s) \rangle \\ &= \langle F(s, x_2, \dots, x_k), w_i(s) \rangle G(x_{>k}) \\ &= H_i(x_2, \dots, x_k)G(x_{>k}), \end{aligned}$$

where $H_i(x_2, \dots, x_k) = \langle F(s, x_2, \dots, x_k), w_i(s) \rangle$ is also a vector-valued function consisting of r_k functions such that

$$(4.8) \quad H_i[j](x_2, \dots, x_k) = \langle F[j](s, x_2, \dots, x_k), w_i(s) \rangle \quad j = 1 \dots r_k$$

Therefore we have shown that unfolding k of $v^{(1)}$ is a rank r_k function, i.e.

$$(4.9) \quad \text{rank} \left[(v_i^{(1)})^k \right] < r_k$$

Furthermore we can see that if we create a matrix-valued function whose rows are H_i , $\mathcal{H} = [H_1, H_2, \dots, H_{r_1}]$ then the quasimatrix V_1 is also extended SVD rank r_k

$$(4.10) \quad V_1(x_{>1})^k = \mathcal{H}(x_2, \dots, x_k)G(x_{>k})$$

Now we can proceed recursively and use the extended SVD of a quasimatrix to separate x_2 from the other dimensions $x_{>2}$ in V_1

$$(4.11) \quad V_1^2(x_{>1}) = V_1(x_2, x_{>2}) = U_2(x_2)V_2(x_{>2}),$$

where U_2 is a $r_1 \times r_2$ matrix-valued function with orthogonal columns and V_2 is a quasimatrix consisting r_2 orthonormal functions. From here we see that $\mathcal{F}_1 = U_1$, $\mathcal{F}_2 = U_2$, and we can proceed with V_2 as we did on V_1 to obtain the rest of the cores. \square

4.1.2. Approximately low-rank case. We continue by showing that if the separated functions f^k of f are *approximately* low-rank (see Equation (3.7)), then we can obtain a FT whose error may be bounded.

THEOREM 4.2. *Suppose that the unfoldings f^k of the function f satisfy*

$$(4.12) \quad f^k = g_k + e_k, \quad \text{rank } g_k = r_k, \quad \|e_k\|_{L_2} = \varepsilon_k, \quad k = 1, \dots, d-1$$

Then a rank $\mathbf{r} = [r_0, r_1, \dots, r_d]$ approximation \hat{f} of f in the the FT-format may obtained with bounded error

$$(4.13) \quad \sqrt{\int (f - \hat{f})^2 dx} \leq \sqrt{\sum_{k=1}^{d-1} \varepsilon_k^2}$$

Proof. The proof is by induction. The case for $d = 2$ follows from the definition of the functional SVD. Now we consider the case for $d > 2$, the first unfolding can be decomposed into

$$f^1(x_1, x_{>1}) = U_1(x_1)V_1(x_{>1})$$

where the columns of U_1 consists of the first r left singular functions of the SVD decomposition of f^1 , $u_i^{(1)}$ and that for each left singular function we have $\langle u_i^{(1)}(s), e(s, x_{>1}) \rangle = 0$ for $i = 1 \dots r_1$ according to the properties of the SVD. Now we can treat each function $v_i^{(1)}$ in V_1 as a $(d-1)$ dimensional tensor which we will approximate by some other tensor $\hat{V}_1 = [v_1^{(1)} \dots v_{r_1}^{(1)}]$ and obtain a bound on the error

$$\begin{aligned} \int (f - \hat{f})^2 dx &= \int (f^1 - U_1 \hat{V}_1)^2 dx \\ &= \int (f^1 - U_1 (\hat{V}_1 + V_1 - V_1))^2 dx \\ &\leq \int (f^1 - U_1 V_1)^2 dx + \int (U_1 (V_1 - \hat{V}_1))^2 dx \\ &\leq \varepsilon_1^2 + \langle V_1 - \hat{V}_1, V_1 - \hat{V}_1 \rangle_Q \end{aligned}$$

Using similar arguments to those in 4.1 we can show that the unfoldings of V_1 are also extended SVD rank r_k within ε_k . Using this information we can find the extended SVD of \hat{V}_1

$$(4.14) \quad \hat{V}_1(x_2, x_{>2}) = U_2(x_2)V_2(x_{>2}),$$

where we will approximate the functions in V_2 to obtain \hat{V}_2

We can proceed to bound the error in a similar fashion

$$\begin{aligned} \langle V_1 - \hat{V}_1, V_1 - \hat{V}_1 \rangle_Q &= \langle V_1 - U_2 \hat{V}_2, V_1 - U_2 \hat{V}_2 \rangle_Q \\ &= \langle V_1 - U_2 (\hat{V}_2 + V_2 - V_2), V_1 - U_2 (\hat{V}_2 + V_2 - V_2) \rangle_Q \\ &\leq \varepsilon_2 + \langle V_2 - \hat{V}_2, V_2 - \hat{V}_2 \rangle_Q \end{aligned}$$

We can then proceed by induction to finally obtain

$$\langle V_1 - \hat{V}_1, V_1 - \hat{V}_1 \rangle_Q \leq \sum_{k=2}^{d-1} \varepsilon_k^2$$

to complete the proof. \square

4.2. Computation. The above two proofs are constructive in the sense that they describe an algorithm which may be used to obtain the FT approximation. In particular, the algorithm requires taking the SVD of sequential vector-valued functions V_i to obtain the next core. However, taking an SVD of such high-dimensional functions is infeasible and in [23] the skeleton decomposition is proposed to replace the SVD as a low-rank representation of each V_i . We now extend the cross-approximation algorithm to the continuous case for obtaining a FT approximation of a black-box function. Then we describe tensor rounding and show how it can be used for rank adaptation in situations for which the rank is not known.

4.2.1. Cross-approximation of multidimensional functions. The cross-approximation of multidimensional functions is a straight forward extension of the cross-approximation of vector-valued functions of a k -separated view of a function described in Section 3.3. The idea is that

whenever the SVD of a vector-valued function is required, a cross-approximation algorithm is performed to obtain a CUR decomposition instead. In order to obtain the maximum volume for each core, we sweep forwards and backwards through each dimension instead of just through the row and column fibers of a *particular* unfolding as described in Section 3.3.

Furthermore, in the high dimensional setting the skeleton decomposition is defined by more than just a single pair of fibers α and y as in Section 3. Instead, we have a set of pairs $\mathbf{I}_x = \{(\mathbf{x}^1, \mathbf{y}^1), \dots, (\mathbf{x}^d, \mathbf{y}^d)\}$, where each pair $(\mathbf{x}^k, \mathbf{y}^k)$ corresponds to the fibers used in the skeleton decomposition of unfolding matrix f^k . For example if we are considering a rank r_2 unfolding matrix $f^2 = f(x_{\leq 2}, x_{>2})$ then we have

$$\mathbf{x}^2 = \left\{ \left(x_1^{(i)}, x_2^{(i)} \right), \dots, \left(x_1^{(r_2)}, x_2^{(r_2)} \right) \right\}$$

and

$$\mathbf{y}^2 = \left\{ \left(x_3^{(i)}, x_4^{(i)}, \dots, x_d^{(i)} \right), \dots, \left(x_3^{(r_2)}, x_4^{(r_2)}, \dots, x_d^{(r_2)} \right) \right\},$$

such that we can write

$$\begin{aligned} f^2(x) &= f(\{x_1, x_2\}, \mathbf{y}^2) \mathbf{G}_2 f(\mathbf{x}^2, \{x_3, x_4, \dots, x_d\}) \\ &= U_2(x_1, x_2) \mathbf{G}_2 V_2(x_3, x_4, \dots, x_d), \end{aligned}$$

where U_2 is a vector-valued function consisting of r_2 two-dimensional functions of x_1 and x_2 and V_2 is a vector-valued function consisting of r_2 $d - 2$ dimensional functions of $x_3 \dots x_d$. As with the cross approximation algorithm in [23] we enforce a nestedness condition on \mathbf{x}^k that means if $(x_1, \dots, x_k, x_{k+1}) \in \mathbf{x}^{k+1}$ then $(x_1, \dots, x_k) \in \mathbf{x}^k$.

Suppose we have a function f whose unfolding functions have functional SVD ranks (2.6) $r_0 = 1$, $r_k = r$ for $k = 1 \dots d - 1$, and $r_d = 1$, we start with the skeleton decomposition of the first unfolding matrix

$$\begin{aligned} f^1(x_1, x_{>1}) &= f(x_1, \mathbf{y}^1) \mathbf{G}_1 f(\mathbf{x}^1, x_{>1}) \\ &= \mathcal{F}_1(x_1) V_1(x_{>1}), \end{aligned}$$

where $\mathcal{F}_1(x_1) = f(x_1, \mathbf{y}^1) \mathbf{G}_1$ and V_1 is a vector-valued function such that $V_1[i](x_{>1}) = f(x_1^{(i)}, x_{>1})$ where $x_1^{(i)}$ is the i -th element of \mathbf{x}^1 . Now we have V_1 is an extended SVD rank r vector-valued function, thus we can find its skeleton decomposition

$$\begin{aligned} V_1(x_{>1}) &= V_1(x_2, x_{>2}) \\ &= f(\mathbf{x}^1, x_2, x_{>2}) \\ &= f(\mathbf{x}^1, x_2, \mathbf{y}^2) \mathbf{G}_2 f(\mathbf{x}^2, x_{>2}), \end{aligned}$$

where the second equality is the definition of V_1 and the third equality comes from the definition of the skeleton decomposition and the fact that \mathbf{x}^1 and \mathbf{x}^2 are nested. From the third equality we can denote $\mathcal{F}_2(x_2) = f(\mathbf{x}^1, x_2, \mathbf{y}^2) \mathbf{G}_2$ and $V_2 = f(\mathbf{x}^2, x_{>2})$. We now have the first two cores \mathcal{F}_1 and \mathcal{F}_2 and can proceed recursively with V_2 . Once we finish through the forward sweep, we can perform a similar backwards sweep. This entire iteration may be performed until convergence.

4.2.2. Rounding and rank adaptation. The cross-approximation procedure discussed above relies on specifying the ranks r_k a-priori. In practice, one can find the ranks adaptively through a procedure called “rounding”. The idea is that if a FT of certain ranks can be well approximated by an FT of smaller ranks, then we have overestimated the ranks we have used in the cross-approximation algorithm and can be confident about the results. We now describe a continuous extension to the rounding algorithm.

Rounding deals with generating and ϵ -level approximation of a FT. This operation is often necessary because the ranks of an FT may be higher than necessary depending on how it was created. The rounding procedure for FT turns out to be very similar to the rounding procedure for TT [22], with the primary difference being the notions of left and right orthogonality.

The rounding procedure follows directly from the definition of the ranks of the separated view of the function f . Suppose we start with the first unfolding of a function in tensor train format f

$$\begin{aligned} f^1(x_1, x_{>1}) &= \mathcal{F}_1(x_1) [\mathcal{F}_2(x_2) \mathcal{F}_3(x_3) \dots \mathcal{F}_d(x_d)] \\ &= F_1 V_1 \end{aligned}$$

where we have treated the first core \mathcal{F}_1 as a vector-valued function, and therefore denote it as F_1 . Furthermore we can treat the products of cores $2 \dots d$ as a vector-valued function denoted by V_1 . Using this expression we can see that f^1 is written as a rank- r_1 function. Suppose that we would like to further compress f^1 through a reduced SVD. To reduced SVD can be obtained by first performing two QR decompositions: $F_1 = Q_1 \mathbf{R}_1$ and $V_1 = Q_2 \mathbf{R}_2$, and then taking the reduced SVD: $\mathbf{R}_1 \mathbf{R}_2^T = \mathbf{X} \mathbf{D} \mathbf{Y}$, where \mathbf{D} is a diagonal matrix whose values are all greater than the truncation level δ . Then we have the following adjustments to the cores of the rounded FT \hat{f} . Following these two approximations, we can compute new cores $\hat{\mathcal{F}}_1$ and $\hat{\mathcal{F}}_2$

$$(4.15) \quad \hat{\mathcal{F}}_1 \equiv \hat{F}_1 = Q_1$$

$$(4.16) \quad \hat{V}_1 = \mathbf{X} \mathbf{D} \mathbf{Y} V_1 \Rightarrow \hat{\mathcal{F}}_2(x_2) = \mathbf{X} \mathbf{D} \mathbf{Y} \mathcal{F}_2(x_2),$$

Now we have reduced rank r_1 to \hat{r}_1 and move on to the second unfolding matrix; however, we use the updated cores. In this case we have

$$\begin{aligned} f^2(x_{\leq 2}, x_{>2}) &= [\hat{\mathcal{F}}_1(x_1) \hat{\mathcal{F}}_2(x_2)] [\mathcal{F}_3(x_3) \dots \mathcal{F}_d(x_d)] \\ &= U_1 V_2. \end{aligned}$$

From this point we can again obtain the reduced SVD and proceed until we have dealt with all of the unfoldings — i.e., obtain $f^3 = U_2 V_3, \dots, f^{d-1} = U_{d-2} V_{d-1}$.

The difficulty with this algorithm as described is obtaining the QR decomposition of V_1 and then the QR decomposition of U_1 . If we assume that at the start of the rounding algorithm all the cores $\mathcal{F}_2, \dots, \mathcal{F}_d$ have *orthonormal rows* then we can show that V_1 already has orthonormal columns, and thus we only have to deal with F_1 .

The concept of *orthonormal rows* is analogous to left orthogonality [22] and, relatedly, *orthonormal columns* is analogous to right orthogonality. The two definitions are explicitly given below where the inner products refer to the inner products between quasimatrices as given in (2.4).

DEFINITION 4.3 (Orthonormal rows). *A function train core \mathcal{F}_k has orthonormal rows if*

$$(4.17) \quad \langle \mathcal{F}_k[i, :](x), \mathcal{F}_k[j, :](x) \rangle = \delta_{i,j} \quad \text{for } i, j = 1 \dots r_{k-1}$$

DEFINITION 4.4 (Orthonormal columns). *A function train core \mathcal{F}_k has orthonormal columns if*

$$(4.18) \quad \langle \mathcal{F}_k[:, i](x), \mathcal{F}_k[:, j](x) \rangle = \delta_{i,j} \quad \text{for } i, j = 1 \dots r_k$$

Now to show that V_1 has orthonormal columns if all the cores except the first one have orthonormal rows, we consider the inner product between two columns of V_1 . Let column k of V_1 be denoted by v_k

$$v_k(x_2, \dots, x_d) = \mathcal{F}_2[k, :](x_2) \mathcal{F}_3(x_3) \dots \mathcal{F}_d(x_d)$$

Now, suppose we compute $\langle v_k, v_l \rangle$

$$\begin{aligned}
\langle v_k, v_l \rangle &= \int v_k(x_2, \dots, x_d) v_l(x_2, \dots, x_d) dx_2 \dots dx_d \\
&= \int (\mathcal{F}_2[k, :](x_2) \mathcal{F}_3(x_2) \dots \mathcal{F}_d(x_d)) (\mathcal{F}_2[l, :](x_2) \mathcal{F}_3(x_2) \dots \mathcal{F}_d(x_d)) dx_2 \dots x_d \\
&= \int (\mathcal{F}_2[k, :](x_2) \mathcal{F}_3(x_2) \dots \mathcal{F}_d(x_d)) (\mathcal{F}_d(x_d)^T \dots \mathcal{F}_3(x_2)^T \mathcal{F}_2[l, :](x_2)^T) dx_2 \dots x_d \\
(4.19) \quad &= \int \mathcal{F}_2[k, :](x_2) \left(\int \mathcal{F}_3(x_3) \left(\dots \left(\int \mathcal{F}_d(x_d) \mathcal{F}_d(x_d)^T dx \right) \dots \right) \mathcal{F}_3(x_3)^T dx_3 \right) \mathcal{F}_2[l, :](x_2)^T dx_2
\end{aligned}$$

Now, because of row orthonormality we have

$$\int \mathcal{F}_k(x_k) \mathcal{F}_k^T(x_k) dx_k = \mathbf{I}_{r_{k-1}},$$

and therefore

$$\langle v_k, v_l \rangle = \int \mathcal{F}_2[k, :](x_2) \mathcal{F}_2[l, :](x_2)^T dx_2 = \delta_{k,l}.$$

Thus, in such a situation we have $\mathbf{R}_2 = \mathbf{I}_{r_1}$.

We now move onto dealing with taking the QR decomposition of U_1 (and by extension U_2, U_3, \dots). Using the same argument as above, orthogonalizing the functions in U_1 requires only orthogonalizing the columns of $\hat{\mathcal{F}}_2$ since $\hat{\mathcal{F}}_1$ is already column orthogonalized after dealing with the first unfolding matrix. This means that once we column orthogonalize all of the functions in \mathcal{F}_2 such that $\mathcal{Q}(x_2)\mathbf{R}_2 = \mathcal{F}_2(x_2)$, we could then simply take the reduced SVD of \mathbf{R}_2 and proceed to the next unfoldings. Since we have defined the inner product between quasimatrices and the columns of each core can be considered as quasimatrices, we can column orthogonalize each function core using householder triangularization. Row orthogonalization proceeds by column orthogonalizing the transposed core.

The entire rounding algorithm is provided by Algorithm 3. It starts with a sweep through cores $k = 2 \dots d$ from the right to the left to make sure they all consist of orthonormal functions. Once this is completed, a left-to-right sweep is performed in which the reduced SVD is performed. Core orthogonalization is performed by taking a sequence of QR decompositions denoted by **qr-mvf**. The reduced SVD algorithm performed by first taking the QR decomposition followed by a reduced SVD of \mathbf{R} and is denoted by **svd-core**.

The overall approximation algorithm with rank adaptation can then be implemented by successively increasing, or “kicking” the FT ranks higher until rounding leads to a reduction of all ranks. This type of algorithm ensures that the FT ranks are *overestimated* for the cross-procedure. The pseudocode for the rank-adaptation procedure is provided by Algorithm 4.

5. Continuous multilinear algebra with the FT decomposition. One of the main advantages of working with functions in FT format is their easy integration with multilinear algebra concepts. The operations of addition, multiplication, integration, and inner products may be easily performed for functions in FT format. Addition and multiplication of two functions are performed exactly the same way as for tensor-trains. For **addition** we have the cores of $g(x) = f(x) + a(x)$ to be

$$\mathcal{G}_1(x) = [\mathcal{F}_1(x) \quad \mathcal{A}_1(x)], \quad \mathcal{G}_k(x) = \begin{bmatrix} \mathcal{F}_k(x) & \mathbf{0} \\ \mathbf{0} & \mathcal{A}_k(x) \end{bmatrix}, \quad \mathcal{G}_d(x) = \begin{bmatrix} \mathcal{F}_d(x) \\ \mathcal{A}_d(x) \end{bmatrix},$$

for $k = 2 \dots d$.

Algorithm 3 ft-round: Function-train rounding

Require: An FT f ; accuracy parameter ϵ

Ensure: \hat{f} with reduced ranks such that $\sqrt{\int (f - \hat{f})^2 dx} \leq \epsilon \sqrt{\int f^2 dx}$

```
1:  $\delta = \frac{\epsilon}{\sqrt{d-1}} \sqrt{\int f^2 dx}$ 
2:  $\hat{\mathcal{F}}_d = \mathcal{F}_d$ 
3: for  $k = d$  to 2 do
4:    $\hat{\mathcal{F}}_k(x_k)\mathbf{R} = \text{qr-mvf}(\hat{\mathcal{F}}_k^T)$  # Orthonormalize all elements of core  $k$ 
5:    $\hat{\mathcal{F}}_{k-1} = \mathcal{F}_{k-1}\mathbf{R}^T$ 
6: end for
7: for  $k = 1$  to  $d - 1$  do
8:    $\hat{\mathcal{F}}_k\mathbf{\Lambda}\mathbf{V} = \text{svd-core}(\hat{\mathcal{F}}_k, \delta)$ 
9:    $\hat{\mathcal{F}}_{k+1} = \mathbf{\Lambda}\mathbf{V}\hat{\mathcal{F}}_k$ 
10: end for
```

Algorithm 4 ft-rankadapt: Function-train approximation with rank adaptation

Require: A d -dimensional black-box function $f : \mathcal{X} \rightarrow \mathbb{R}$; Cross-approximation tolerance δ_{cross} ; Number of ranks to increase with each adaptation **kickrank**; Rounding accuracy ϵ_{round} ; Initial ranks \mathbf{r}

Ensure: Approximation \hat{f} such that a rank increase and rounding does not change ranks.

```
1:  $\hat{f} = \text{cross-approx}(f, \mathbf{r}, \delta_{\text{cross}})$ 
2:  $\hat{f}_r = \text{ft-round}(\hat{f}, \epsilon_{\text{round}})$ 
3:  $\hat{\mathbf{r}} = \text{ranks}(\hat{f}_r)$ 
4: while  $\hat{\mathbf{r}} \neq \mathbf{r}$  do
5:   for  $k = 1 \rightarrow d - 1$  do
6:      $\mathbf{r}_k = \hat{\mathbf{r}}_k + \text{kickrank}$ 
7:      $\hat{f} = \text{cross-approx}(f, \mathbf{r}, \delta_{\text{cross}})$ 
8:   end for
9:    $\hat{f}_r = \text{ft-round}(\hat{f}, \epsilon_{\text{round}})$ 
10:   $\hat{\mathbf{r}} = \text{ranks}(\hat{f}_r)$ 
11: end while
12:  $\hat{f} = \hat{f}_r$ 
```

For **multiplication**, $g(x) = f(x)a(x)$ we have

$$\mathcal{G}_k(x) = \mathcal{F}_k(x) \otimes \mathcal{A}_k(x) \quad \text{for } k = 1 \dots d.$$

Differentiation requires differentiating individual cores. For example, consider the partial derivative of a d dimensional function f

$$\frac{df}{dx_k} = \mathcal{F}_1 \dots \mathcal{F}_{k-1} \begin{bmatrix} \frac{d\hat{f}_k^{(1,1)}}{dx_k} & \dots & \frac{d\hat{f}_k^{(1,r_k)}}{dx_k} \\ \vdots & & \vdots \\ \frac{d\hat{f}_k^{(r_{k-1},1)}}{dx_k} & \dots & \frac{d\hat{f}_k^{(r_{k-1},r_k)}}{dx_k} \end{bmatrix} \mathcal{F}_{k+1} \dots \mathcal{F}_d$$

Integration involves integrating over all the one dimensional functions in each core and then per-

forming matrix-vector multiplication $d - 1$ times

$$\begin{aligned}
\int f(x)dx &= \int \mathcal{F}_1(x_1)\mathcal{F}_2(x_2)\dots\mathcal{F}_d(x_d)dx_1\dots dx_d \\
&= \left(\int \mathcal{F}_1(x_1)dx_1\right)\left(\int \mathcal{F}_2(x_2)dx_2\right)\dots\left(\int \mathcal{F}_d(x_d)dx_d\right) \\
(5.1) \quad &= \mathbf{\Gamma}_1\mathbf{\Gamma}_2\dots\mathbf{\Gamma}_d
\end{aligned}$$

where $\mathbf{\Gamma}_k = \int \mathcal{F}_k(x_k)dx_k$ contains entries $\mathbf{\Gamma}_k[i, j] = \int \hat{f}_k^{(i, j)}(x_k)dx_k$.

The **inner product** of two functions is another important operation which is used ubiquitously. Naively, the inner product can be implemented by first computing the product $g(x) = f(x)p(x)$ and then integrating $g(x)$ requiring $\mathcal{O}(dr^4)$ operations. However, this operation can be made more efficient by combining the operations needed for integration and multiplication. For example, Algorithm 5 uses an efficient computation of $v^T(\mathbf{A} \otimes \mathbf{B})$ to perform the inner product in $\mathcal{O}(dr^3)$.

Algorithm 5 ft-inner: Inner product between two functions in FT format

Require: Functions f with ranks $r_k^{(f)}$ and g with ranks $r_k^{(g)}$ in FT format

Ensure: $y = \int f(x)g(x)dx$

- 1: $\mathbf{y} = \int \mathcal{G}_1(x_1) \otimes \mathcal{F}_1(x_1)dx_1$
 - 2: **for** $k = 2$ **to** d **do**
 - 3: $\mathbf{Y} = \text{reshape}(\mathbf{y}, r_{k-1}^{(f)}, r_{k-1}^{(g)})$
 - 4: $\mathcal{T} = \mathcal{F}_k(x)^T \mathbf{Y}$
 - 5: $\mathcal{A} = \mathcal{G}_k(x)^T \mathcal{T}^T(x)$
 - 6: $\mathbf{Y} = \int \mathcal{A}(x_k)dx_k$
 - 7: $\mathbf{y} = \text{reshape}(\mathbf{Y}, 1, r_k^{(f)}r_k^{(g)})$
 - 8: **end for**
 - 9: $y = \mathbf{y}[1]$
-

Furthermore, once in FT format many familiar operators may be applied to a function with relative ease. Consider the Laplacian $\nabla f(x) = g(x) = \sum_{k=1}^d \frac{d^2 f(x)}{dx_k^2}$. Written in this form, one can consider the Laplacian as the summation of d functions $g_k(x)$ in function-train format where

$$g_k(x) = \frac{d^2 f(x)}{dx_k^2}$$

One can imagine using the ability to apply this operator directly on the functional format to construct iterative linear system solvers. In the context of low rank tensor decompositions, these types of solvers are the subject of much current research [18, 30, 9]

6. Implementation. We now turn to the practical issues surrounding implementation of the algorithms described. In particular, we focus on the notion of representing each function in the FT core as a one dimensional function in a known basis, and we will show how this idea is used in the construction of an approximation through the cross approximation algorithm and for FT integration. The ideas are directly transferable to the other multilinear algebra operations.

6.1. One-dimensional fiber adaptation for cross-approximation. We start with cross-approximation. When performing cross-approximation to convert a function to its FT format, one only requires dealing with one dimensional fibers of the function, i.e., we fix all variables except one. These fibers are necessary, for example, in creating R and \mathcal{C} of Algorithm 1. When such operations are required we first *approximate* the fiber in some *basis*. These basis are *adaptive* to each fiber which needs to be approximated. The choice of the approximation can vary from fiber to

fiber. For instance, if one expects the fibers to be smooth functions, then one can use an orthogonal expansion of Legendre polynomials. Alternatively, if one expects or detects local phenomena such as discontinuities, one can turn to piecewise polynomials. Any additional knowledge about the function can be injected in the fiber approximation step. This is a characteristic unique to the FT and doesn't exist for the discrete tensor-train.

To make this idea concrete let us define a routine **approx-fiber** which takes in a one dimensional function and produces an approximation in a given basis. Using one-dimensional fiber approximations we can reproduce Algorithm 1 for the case of a two-dimensional function for illustration purposes.

Algorithm 6 Cross-approximation of a two dimensional function using fiber adaptive approximation

Require: A two-dimensional function $f \in [a, b] \times [c, d]$; Rank estimate r ; Initial y index $\mathbf{y} = [y_1, y_2, \dots, y_r]$; Stopping tolerance $\delta_{cross} > 0$; Approximation scheme **approx-fiber**(f, ϵ_{approx}), Fiber approximation tolerance ϵ_{approx} ;

Ensure: \mathbf{x}, \mathbf{y} such that $\mathbf{F} = f(\mathbf{x}, \mathbf{y})$ has “large” volume

```

1:  $\delta = \delta_{cross} + 1$ 
2:  $f^{(0)} = 0$ 
3:  $k = 1$ 
4: while  $\delta \leq \delta_{cross}$  do
5:   Initialize vector-valued function  $R \in \mathbb{R}^{[a,b] \times r}$ 
6:    $R[k](x) = \text{approx-fiber}(f(x, y_k), \epsilon_{approx})$  for  $k = 1 \dots r$ 
7:    $Q\mathbf{T} = \text{qr-vvf}(R)$ 
8:    $\mathbf{x} = \text{maxvol-mvf}(Q)$ 
9:   Initialize vector-valued function  $C \in \mathbb{R}^{[c,d] \times r}$ 
10:   $C[k](y) = \text{approx-fiber}(f(x_k, y), \epsilon_{approx})$  for  $k = 1 \dots r$ 
11:   $Q\mathbf{T} = \text{qr-vvf}(C)$ 
12:   $\mathbf{y} = \text{maxvol-mvf}(Q)$ 
13:   $\hat{\mathbf{Q}} = [Q(y_1) \ Q(y_2) \ \dots \ Q(y_r)]$ 
14:   $f^{(k)}(x, y) = f(x, \mathbf{y}) \hat{\mathbf{Q}}^\dagger Q(y)$ 
15:   $\delta = ||f^{(k)} - f^{(k-1)}|| / ||f^{(k)}||$ 
16:   $k = k + 1$ 
17: end while
```

Note that we can implement different approximation schemes for different dimensions and even for different fibers within each dimension. Furthermore, once R and C are represented in a known basis, the steps required for the maxvol algorithm can be adapted to the types of approximations. For example Line 5 of Algorithm 2 requires finding the maximum element of a quasimatrix. If each column of the quasimatrix is a Legendre polynomial, then the maximum element in each column can be found through a root finding / eigenvalue procedure to arbitrary precision. Such a procedure removes any need for discretization or gridding! The other methods such as qr-qm and lu-qm can be performed for quasimatrices made up of polynomial expansions just as in [4, 33, 32]. If splines are used, the procedures can be adapted appropriately.

We now demonstrate the advantage of adapting each fiber on two examples where we show that the resulting function evaluations *do not* lie on a tensor product grid. The example shows one of the primary ways this algorithm is different from the tensor-train representation and from other functional tensor train approaches such as [7]. Consider the approximation of two canonical rank-1

functions

$$(6.1) \quad f_{\sin}(x, y) = \sin(10x + \frac{1}{4})(y + 1)$$

$$(6.2) \quad f_{\text{genz2d}}(x, y) = \begin{cases} 0 & \text{if } x > 0.5 \text{ or } y > 0.5 \\ \exp(5x + 5y) & \text{otherwise} \end{cases}$$

where (6.2) is a two dimensional Genz function of the discontinuous family [11].

For (6.1) we use a global expansion of Legendre polynomials ϕ_i to represent each fiber

$$f = \sum_{i=1}^n a_i \phi_i(x), \quad \int \phi_i(x) \phi_j(x) dx = \delta_{i,j},$$

where the coefficients are determined through projection using Clenshaw-Curtis quadrature. The order of the approximation n is progressively increased until four sequential coefficients are $a_i \leq \epsilon_{\text{approx}} = 10^{-10}$. Furthermore, even though (6.1) is a FT-rank 1 function we use a rank-2 approximation in order to help illuminate the difference in function evaluations between our method and existing tensor-based approximation methods.

For (6.2) we can no longer use a global polynomial expansion due to the discontinuity and therefore we turn to a piecewise polynomial approximation. We first use a one dimensional edge detection method based on polynomial annihilation [3, 2, 17] to locate the discontinuities. We then approximate each polynomial piece using the same polynomial expansion scheme described above. Furthermore, we employ a rank-1 approximation of this function.

Figure 2 shows the function and the resulting evaluation locations, where both approximations achieve machine precision accuracy. In the left panel, we see that the function is evaluated densely along the top of the plot and coarsely along the bottom— i.e., the evaluations do not lie on a tensor product grid. In the right panel, we see that the algorithm clusters points around the discontinuity as desired.

Another point to note is that once we have the function-trains of each of these functions we can perform computation with them directly in compressed form. For example, we can now integrate the discontinuous function f_{genz2d} . Such integration cannot be performed using array-based tensor-train algorithms without manually splitting the domain because it would require specialized integration rules to deal with the discontinuity. When representing everything in functional form, we are able to perform the integration and approximation automatically. In Section 7.1.2, we will show integration performance on higher dimensional discontinuous Genz functions.

6.2. Function-train integration. Suppose that we have a FT representation in which each core is made of one-dimensional functions which are Legendre polynomial expansions. In this case, we can integrate each core easily if we know how to integrate Legendre polynomials. More concretely, suppose that the fibers of core k are represented using a Legendre polynomial expansion

$$\hat{f}_k^{(i,j)}(x) = \sum_{\ell=1}^n a_{\ell}^{(k,i,j)} \phi_{\ell}(x), \quad \langle \phi_{\ell}, \phi_m \rangle = 2\delta_{\ell,m}, \quad \phi_1 = 1$$

Then we can compute its integral

$$\int \hat{f}_k^{(i,j)}(x) dx = \int \sum_{\ell=1}^n a_{\ell}^{(k,i,j)} \phi_{\ell}(x) dx = \sum_{\ell=1}^k a_{\ell}^{(k,i,j)} \int \phi_{\ell}(x) dx = 2a_1^{(k,i,j)}$$

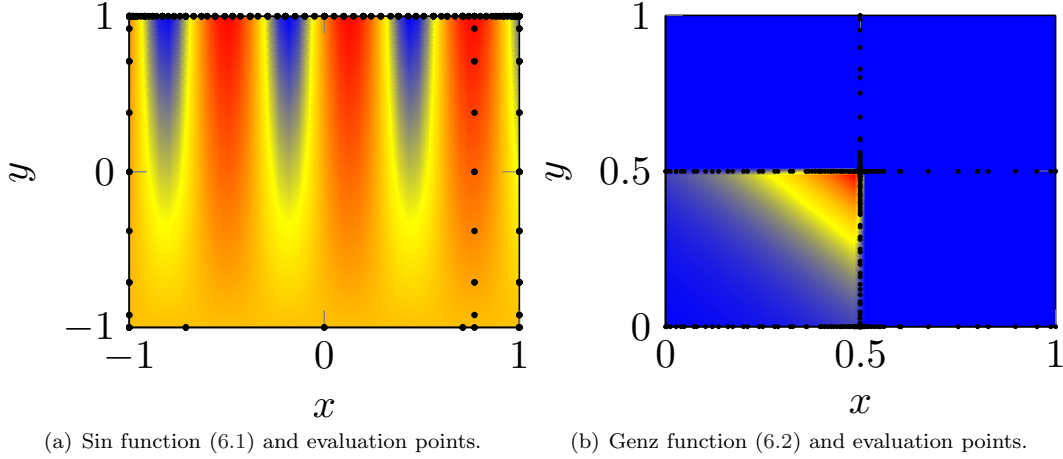


FIG. 2. Contour plots and evaluations of f_{\sin} and f_{genz2d} .

This means that the integral of each core results in the matrix consisting of scaled values of the first coefficient of each function

$$\int \mathcal{F}_k(x_k) dx_k = 2 \begin{bmatrix} a_1^{(k,1,1)} & \dots & a_1^{(k,1,r_k)} \\ \vdots & & \vdots \\ a_1^{(k,r_{k-1},1)} & \dots & a_1^{(k,r_{k-1},r_k)} \end{bmatrix}$$

From here we can use Equation (5.1) to obtain the integral.

7. Results and applications. We now apply the algorithm on a variety of integration and function approximation test problems. Showing the benefits of using the FT for integration highlights its applicability for a wide variety of common problems in computational science. The approximation test problems serve to show the rank-adaptation scheme on a wide variety of benchmark problems. All of the experiments performed in this section were done with the Compressed Continuous Computation (C^3) toolbox [15].

7.1. Integration. We first show the performance of the FT algorithm on two integration examples.

7.1.1. Rank-2 Sin function. Analogously to [23], we consider the FT rank-2 function

$$(7.1) \quad f(x) = \sin(x_1 + x_2 + \dots + x_d)$$

for which we know the analytic integral

$$(7.2) \quad \int_{[0,1]^d} f(x) dx = \text{Im} \left[\left(\frac{e^i - 1}{i} \right)^d \right]$$

We seek to study the performance associated with computing this integral as a function of dimension and fiber adaptation error. Specifically we choose an adaptive procedure based on approximating each fiber as Legendre polynomial series expansion. Each approximation begins with a fifth order expansion, and each time adaptation occurs the expansion order goes from $k \rightarrow 2k - 1$. We stop adaptation after the last *two* coefficients of the expansion drop below a tolerance δ . Figure 3 presents results studying the integration error as a function of this δ and the dimensionality of the problem.

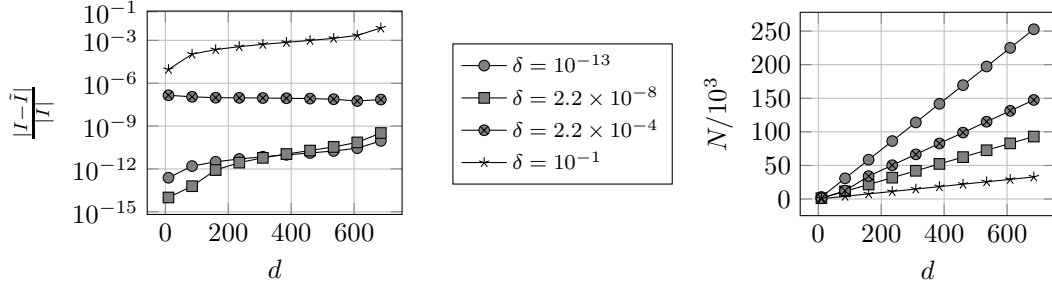


FIG. 3. Errors (left panel) and number of evaluations (right panel) involved in the integration of (7.1) as a function of dimension d and adaptation parameter δ .

7.1.2. Rank-1 discontinuous Genz function. We now demonstrate integration on the Genz discontinuous functions of various dimensions. Specifically, we integrate the function $f : [0, 1]^d \rightarrow \mathbb{R}$ defined as

$$(7.3) \quad f(x_1, x_2, \dots, x_d) = \begin{cases} 0 & \text{if } x_i > \frac{1}{2} \text{ for any } i = 1 \dots d \\ \exp\left(\sum_{i=1}^d 5x_i\right) & \text{otherwise} \end{cases}$$

The analytical integral of (7.3) is

$$I[f] = \left(\frac{\exp\left(\frac{5}{2}\right) - 1}{5} \right)^d,$$

where we see that this problem is quite challenging because the integral grows exponentially with dimension. For example, for 10 dimensions this integral is $I \approx 3.131 \times 10^3$ and for 100 dimensions this integral is $I \approx 9.05455 \times 10^{34}$.

We integrate this function by first converting it to its rank-1 function-train representation. The fibers are approximated by sixth order piecewise polynomials. The discontinuities are located automatically using a polynomial annihilation edge detection routine [3, 2, 17]. After conversion to the function-train representation, we integrate the FT using the technique described in Section 5. The resulting errors and required function evaluations as a function of dimension are depicted in Figure 4.

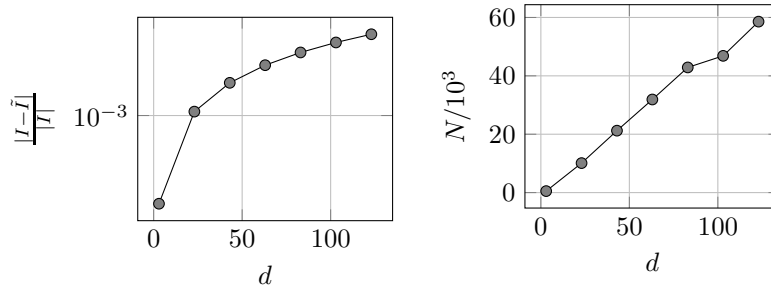


FIG. 4. Errors (left panel) and number of evaluations (right panel) involved in the integration of (7.3) as a function of dimension d .

The results indicate that we are able to achieve $\mathcal{O}(10^{-3})$ relative accuracy for all of the dimensions and that the number of function evaluations required scales linearly with dimension. Furthermore, we are able to approximate large integral values, indicating a general robustness of the algorithm. We also note that such an integration would be extremely difficult to perform using

Fiber approximation	Cross-approximation	Rank adaptation
Initialize to 7th degree Increase degree $k \rightarrow 2k + 1$ $\epsilon_{approx} = 10^{-7}$	$\delta_{cross} = 10^{-3}$ ≤ 5 sweeps	$\epsilon_{round} = 10^{-5}$ kickrank = 5 ≤ 5 adaptations

TABLE 3

Algorithmic parameters used for approximation of the simulation library benchmark problems shown in Table 4.

either the discrete tensor-train or the spectral tensor-train [7] techniques because the discontinuities pose problems for most integration rules. Individual fiber adaptation is critical in order to locate the discontinuities.

7.2. Approximation. We now test the FT approximation on a set of benchmark approximation problems. The first set of test functions we will use are taken from the Emulation/Prediction Test Problems set in [29]. In particular, we have chosen to examine a subset of functions which have more than two input dimensions. Secondly, we explore some of the effects of different algorithm parameter choices for the approximation of a quantity of interest arising from an elliptic pde.

7.2.1. Simulation library benchmark problems. To standarize our algorithm for each benchmark function we fix the algorithmic parameters to those shown in Table 3. In particular, we utilize Legendre polynomial expansions with Gauss-Legendre quadrature for fiber adaptation and we perform rank adaptation using Algorithm 4.

For each problem we normalize each input to $[-1, 1]$ and compute the Relative RMSE error using 10000 Monte carlo samples as

$$error = \sqrt{\frac{\sum_{i=1}^{10000} (f(x_i) - \hat{f}(x_i))^2}{\sum_{i=1}^{10000} f(x_i)^2}}$$

Table 4 show our results. Except for the Robot Arm function, each of the test functions are found to be low rank. The Robot Arm function led to several difficulties including excessive basis adaptation and a non-converged maximum rank after 5 rank increase steps. Previous results in the literature [1] have also found this function to be difficult to approximate using the polynomial basis. The authors of [1] note that a Fourier basis or trigonometric polynomials may be more appropriate for this function to exploit the periodic inputs and oscillatory nature of the function. As is, the FT approximation performs rather poorly on this 8 dimensional problem, requiring almost 3 times as many evaluations as the 7 dimensional Piston function.

7.2.2. Elliptic PDE. We next explore the effects of various parameters of the FT approximation algorithm on a model of sub-surface flow typically encountered in UQ applications. Consider the following one dimensional elliptic partial differential equation

$$(7.4) \quad \frac{\partial}{\partial s} \left(k(s, \omega) \frac{\partial u}{\partial s} \right) = s^2,$$

for $s \in (0, 1)$ with $u(s)|_{s=0} = 0$ and $\frac{\partial u}{\partial s}|_{s=1} = 0$. In the UQ context, we would like to consider the effects of an unknown permeability field $k(s, \omega)$ on some function of the output pressure u . To this end, $k(s, \omega)$ is modelled as a random process, typically with a log-Gaussian distribution:

$$\log [k(s, \omega) - a] \sim \mathcal{N}(0, c(s, s'))$$

where the covariance kernel is $c(s, s') = \sigma^2 \exp(-\frac{|s-s'|}{2l^2})$. The objective is to propagate uncertainty in $k(s, \omega)$ to some quantity of interest $Qoi(u(s))$, where $u(s)$ is the solution of the PDE for a

Name (dimension)	Ranks	Number of evaluations / error	Comments
Borehole (8)	(1,2,2,2,3,3,2,2,1)	8026 / 5.2×10^{-5}	
OTL Circuit (6)	(1,3,2,2,2,2,1)	3341 / 1.5×10^{-4}	
Piston (7)	(1,2,4,4,3,2,2,1)	16368 / 1.0×10^{-2}	
Robot Arm (8)	(1,1,7,21,24,25,21,11,1)	45445 / 9.8×10^{-2}	No 1d basis adaptation
Wing Weight (10)	(1,2,2,2,2,2,2,2,2,1)	8699 / 1.3×10^{-10}	
Friedman (5)	(1,4,2,2,2,1)	1957 / 3.2×10^{-5}	
Gramacy and Lee 2009 (6)	(1,2,3,2,1,1,1)	9520 / 6.7×10^{-3}	
Dette and Pepelyshev (2010) 8-Dimensional (8)	(1,3,2,3,3,3,2,2,1)	6252 / 1.5×10^{-5}	
Dette and Pepelyshev (2010) Exponential (3)	(1,2,2,1)	3903 / 5.2×10^{-8}	

TABLE 4

Performance of the FT approximation algorithm on a set of multi-dimensional test functions from [29].

realization of the permeability field $k(s)$. For simplicity, we will consider $Qoi(u(s)) = u(0.7)$. In order to obtain a finite dimensional representation of $k(s, \omega)$ we use the Karhunen-Loève expansion to express the random field as a weighted sum of the eigenfunctions of c . In particular if we compute the eigenfunctions and eigenvalues of the correlation function $\int c(s, s') \phi_i(s') ds' = \lambda_i \phi_i(s)$ then any realization of this Gaussian process may be represented as $k(s, \omega) = \sum_{i=1}^{\infty} \sqrt{\lambda_i} \phi_i(s) \xi_i(\omega)$, where $\xi_i(\omega)$ are now Gaussian random variables. We approximate these eigenfunctions and eigenvalues on a 100 point discretized grid and truncate the expression after 24 modes. We will perform the approximation for various 3 different parameters (a, σ^2, l) indicating various difficulties of the problem.

In particular, we will investigate 3 problems corresponding to an “easy” problem $(a, \sigma^2, l) = (0.0, 0.1, 0.25)$, a “moderate” difficulty problem $(a, \sigma^2, l) = (0.5, 1.0, 0.15)$ and a more difficult problem $(a, \sigma^2, l) = (0.0, 1.0, 0.15)$.

We now seek to build an FT approximation of 24 dimensions to map from the KL modes ξ_i to the quantity of interest. In order to still use Legendre polynomial expansions to represent our one dimensional fibers we need to reparameterize the problem to one with uniform random variables on $[0, 1]$. We can do this using the inverse CDF of a Gaussian random variable to make new variables $\hat{\xi}_i = \Phi^{-1}(\xi_i)$, where Φ^{-1} is the inverse CDF $\hat{\xi}_i$ are the new, uniformly distributed, random variables. Now the approximation is constructed from $(\hat{\xi}_1, \hat{\xi}_2, \dots, \hat{\xi}_{24}) \rightarrow u(0.7)$.

Our experiments investigate how the error, number of function evaluations and maximum rank change with ϵ_{approx} and ϵ_{round} . We fix the cross parameters to $\delta_{cross} = 10^{-3}$ and a maximum 3 sweeps. The rank adaptation parameters are fixed at `kickrank` = 5 and maximum 4 rank adaptations. Finally, we initialize the fiber approximations with a second order polynomial expansions. The results for the three problem setups described above are provided in Figure refig:ellipres. We can see a general trend of decreasing error with increased evaluations; however, there is a wide scatter of errors for a given N . The reason for this scatter can be seen from the plots in Figure 5.

Several patterns are apparent in the results of Figure 5. We first consider the ranks for each problem, shown in the third column. We immediately see that the approximation accuracy practically does not impact the maximum rank found by the rank adaptation. The maximum rank

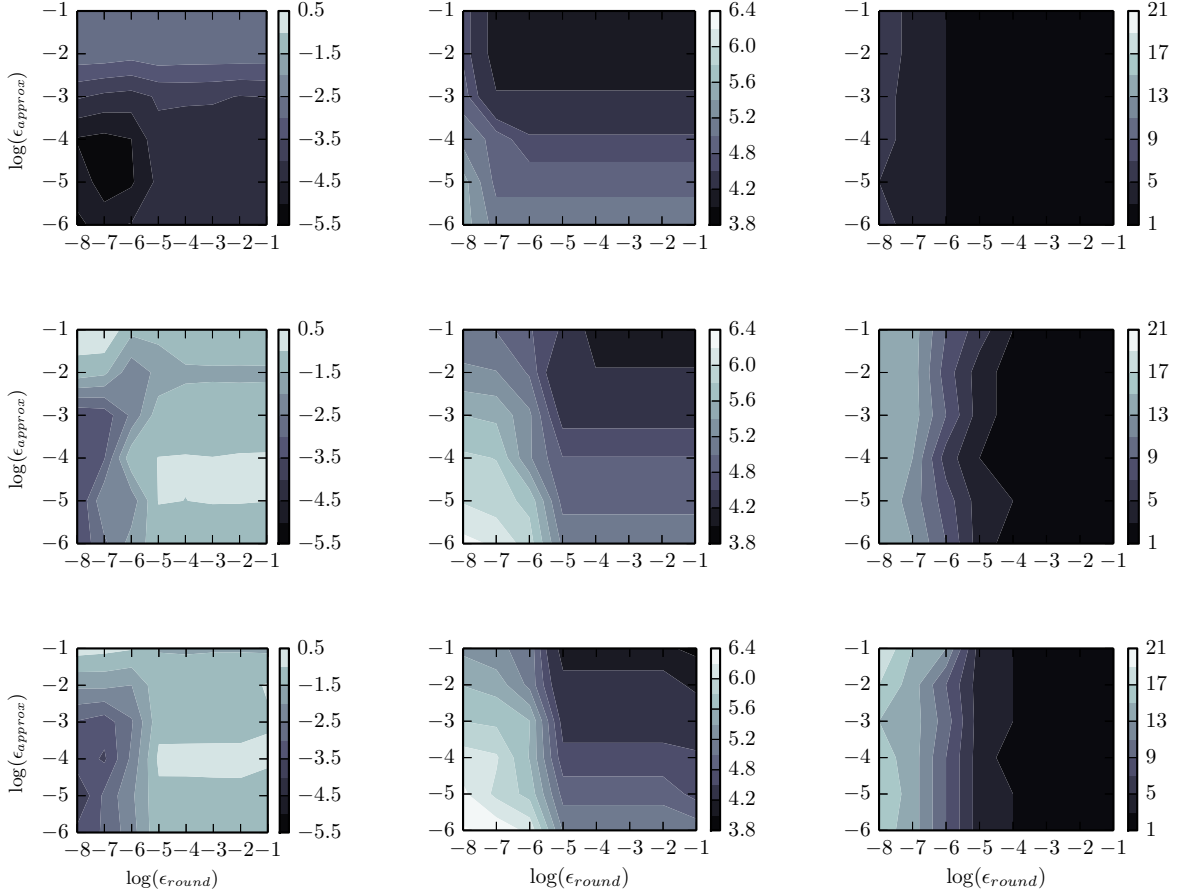


FIG. 5. $\log(\text{error}^2)$ (left column), $\log(\text{number of evaluations})$ (middle column), and maximum rank (right column) for three different configurations of the elliptic problem (7.4) corresponding to different combinations of (a, σ^2, l) . In particular, the top row corresponds to $(0.0, 0.1, 0.25)$, middle row corresponds to $(0.5, 1.0, 0.15)$, and the bottom row corresponds to $(0.0, 1.0, 0.15)$.

found by the adaptation only changes as rounding tolerance decreases. Furthermore, we see that the rank increases as we move down the table. This corresponds to the notion of the difficulty of each problem, and is analogous to the eigenvalue decay. Furthermore, the rank is higher for the last problem showing that the problem is more difficult when $a = 0$ rather than $a = 0.5$. One further concept is that unlike in array-based tensor-train algorithms or the spectral tensor-train algorithm, the maximum rank is *not bounded* by any number of function evaluations. Rounding is critical to restricting the growth of the rank.

The number of function evaluations results shown in column 2 also display some interesting properties. In particular, for each of the three models, there are two separate regimes of patterns. Before the rounding tolerance becomes tight enough so that the rank increases, the number of function evaluations is unaffected by the tolerance. The number of evaluations is only affected by the fiber approximation accuracy. This makes sense because the number of function evaluations is proportional to $\mathcal{O}(n d r^2)$, where n can be thought of as the average function evaluations for each fiber, and in this regime, the rank is constant. Once the rank starts decreasing we see that the number of function evaluations grows with both tighter approximation and rounding tolerances.

However, the changes in the number of evaluations change more rapidly with increasing rounding tolerance because reducing the rounding tolerance results in a rapid increase in rank.

Finally, column 1 shows $\log(\text{error}^2)$ exhibits a similar pattern as the number of function evaluations. In particular, the error is fairly constant until the rank starts decreasing. Once the rank starts decreasing both approximation tolerance and rounding tolerance affect the error. Furthermore, if we fix the rounding tolerance and decrease the approximation tolerance we see a rapid change in approximate error followed by a relatively large plateau area, suggesting that beyond a given ϵ_{approx} the fiber approximation accuracy can no longer increase. Overall, these results suggest that it is possible to find a reasonable value for the rank before increasing the fiber approximation accuracy. Future work will require investigating how to adapt these two parameters in the best way.

8. Conclusion. We have developed a function approximation technique by extending the tensor-train decomposition using continuous linear algebra. Several algorithms were created to enable this extension: a maxvol algorithm for obtaining a dominant sub-matrix from a matrix-valued function, a cross-approximation algorithm for obtaining the skeleton decomposition of a vector-valued function, a cross-approximation algorithm for representing a multivariate function in FT format, and a rounding algorithm for reapproximating a FT with one of lower ranks.

Constructing the algorithm using continuous linear algebra provides a flexible way for incorporating and exploiting more than just low-rank structure. For example, representing the univariate functions making up the cores of the FT with a polynomial approximation allows us to exploit function regularity. This characteristic is important for problems, such as the solution of differential equations, that traditionally require discretization of high-dimensional functions. These resulting solutions are typically sensitive to the choice of discretization, and the problem of adapting the discretization to capture local features is critical for achieving high accuracy. Our continuous representation automatically includes adaptation to local features and does not require the specification of a tensor product set of candidate evaluation locations.

Our framework also enables polynomial time performance for various multidimensional linear algebra algorithms applied to low-rank high dimensional problems. We anticipate that these tools will be useful for the design of many algorithms. In the context of the solution of partial differential equations [21] provides a method for transitioning away from the traditional *discretize-then-solve* methodology, and future work will aim at extending these methods using the FT. Future work will also be targeted towards exploiting low-rank structure in inference and control. In particular, we would like to modify the optimal stochastic control framework described in [16] to avoid state space discretization altogether. Future algorithmic extensions will require the incorporation of `dmrg-cross` [25] to develop rank-revealing algorithms. These rank-revealing algorithms show promise for being more efficient for rank estimation than the rank-adaptation algorithm described here.

Acknowledgments. This work was supported by the National Science Foundation through grant IIS-1452019.

REFERENCES

- [1] J. AN AND A. OWEN, *Quasi-regression*, Journal of complexity, 17 (2001), pp. 588–607.
- [2] R. ARCHIBALD, A. GELB, R. SAXENA, AND D. XIU, *Discontinuity detection in multivariate space for stochastic simulations*, Journal of Computational Physics, 228 (2009), pp. 2676–2689.
- [3] R. ARCHIBALD, A. GELB, AND J. YOON, *Polynomial fitting for edge detection in irregularly sampled signals and images*, SIAM Journal on Numerical Analysis, 43 (2005), pp. 259–279.
- [4] Z. BATTLES AND L. N. TREFETHEN, *An extension of matlab to continuous functions and operators*, SIAM Journal on Scientific Computing, 25 (2004), pp. 1743–1770.
- [5] M. BEBENDORF, *Approximation of boundary element matrices*, Numerische Mathematik, 86 (2000), pp. 565–589.
- [6] ———, *Adaptive cross approximation of multivariate functions*, Constructive approximation, 34 (2011), pp. 149–179.

- [7] D. BIGONI, A. ENGSIG-KARUP, AND Y. MARZOUK, *Spectral tensor-train decomposition*, arXiv preprint arXiv:1405.5713, (2014).
- [8] C. BOUTSIDIS AND D. P. WOODRUFF, *Optimal cur matrix decompositions*, in Proceedings of the 46th Annual ACM Symposium on Theory of Computing, ACM, 2014, pp. 353–362.
- [9] S. DOLGOV, *Tt-gmres: solution to a linear system in the structured tensor format*, Russian Journal of Numerical Analysis and Mathematical Modelling, 28 (2013), pp. 149–172.
- [10] P. DRINEAS, R. KANNAN, AND M. W. MAHONEY, *Fast monte carlo algorithms for matrices ii: Computing a low-rank approximation to a matrix*, SIAM Journal on Computing, 36 (2006), pp. 158–183.
- [11] A. GENZ, *Testing multidimensional integration routines*, in Proc. of international conference on Tools, methods and languages for scientific and engineering computation, Elsevier North-Holland, Inc., 1984, pp. 81–94.
- [12] S. GOREINOV, I. V. OSELEDETS, D. V. SAVOSTYANOV, E. E. TYRTYSHNIKOV, AND N. L. ZAMARASHKIN, *How to find a good submatrix*, Matrix methods: theory, algorithms and applications, (2010), p. 247.
- [13] S. A. GOREINOV AND E. E. TYRTYSHNIKOV, *The maximal-volume concept in approximation by low-rank matrices*, Contemporary Mathematics, 280 (2001), pp. 47–52.
- [14] S. A. GOREINOV, N. L. ZAMARASHKIN, AND E. E. TYRTYSHNIKOV, *Pseudo-skeleton approximations by matrices of maximal volume*, Mathematical Notes, 62 (1997), pp. 515–519.
- [15] A. GORODETSKY, *C³: Compressed Continuous Computation library*. <https://github.com/goroda/Compressed-Continuous-Computation>, 2014-2015.
- [16] A. GORODETSKY, S. KARAMAN, AND Y. MARZOUK, *Efficient high-dimensional stochastic optimal motion control using tensor-train decomposition*, in Proceedings of Robotics: Science and Systems, Rome, Italy, July 2015.
- [17] A. GORODETSKY AND Y. MARZOUK, *Efficient localization of discontinuities in complex computational simulations*, SIAM Journal on Scientific Computing, 36 (2014), pp. A2584–A2610.
- [18] B. KHOROMSKIY AND I. OSELEDETS, *Qtt approximation of elliptic solution operators in higher dimensions*, Russian Journal of Numerical Analysis and Mathematical Modelling, 26 (2011), pp. 303–322.
- [19] M. MAHONEY AND P. DRINEAS, *Cur matrix decompositions for improved data analysis*, Proceedings of the National Academy of Sciences, 106 (2009), pp. 697–702.
- [20] G. N. NEWSAM, *On the asymptotic distribution of the eigenvalues of discretizations of a compact operator*, in Proc. Center for Mathematical Analysis, vol. 17, 1988, pp. 92–105.
- [21] S. OLVER AND A. TOWNSEND, *A practical framework for infinite-dimensional linear algebra*, in Proceedings of the 1st First Workshop for High Performance Technical Computing in Dynamic Languages, IEEE Press, 2014, pp. 57–62.
- [22] I. V. OSELEDETS, *Tensor-train decomposition*, SIAM Journal on Scientific Computing, 33 (2011), pp. 2295–2317.
- [23] I. V. OSELEDETS AND E. E. TYRTYSHNIKOV, *Tt-cross approximation for multidimensional arrays*, Linear Algebra and its Applications, 432 (2010), pp. 70–88.
- [24] R. B. PLATTE AND L. N. TREFETHEN, *Chebfun: a new kind of numerical computing*, in Progress in Industrial Mathematics at ECMI 2008, Springer, 2010, pp. 69–87.
- [25] D. SAVOSTYANOV AND I. OSELEDETS, *Fast adaptive interpolation of multi-dimensional arrays in tensor train format*, in Multidimensional (nD) Systems (nDs), 2011 7th International Workshop on, IEEE, 2011, pp. 1–8.
- [26] D. V. SAVOSTYANOV, *Quasioptimality of maximum-volume cross interpolation of tensors*, Linear Algebra and its Applications, 458 (2014), pp. 217–244.
- [27] E. SCHMIDT, *Zur theorie der linearen und nicht linearen integralgleichungen zweite abhandlung*, Mathematische Annalen, 64 (1907), pp. 161–174.
- [28] G. W. STEWART, *Fredholm, hilbert, schmidt: three fundamental papers on integral equations*, See www.cs.umd.edu/~stewart/FHS.pdf, (2011).
- [29] S. SURJANOVIC AND D. BINGHAM, *Virtual library of simulation experiments: Test functions and datasets*. Retrieved September 4, 2015, from <http://www.sfu.ca/~ssurjano>.
- [30] C. TOBLER, *Low-rank tensor methods for linear systems and eigenvalue problems*, PhD thesis, ETH Zürich, 2012.
- [31] A. TOWNSEND AND L. N. TREFETHEN, *An extension of chebfun to two dimensions*, SIAM Journal on Scientific Computing, 35 (2013), pp. C495–C518.
- [32] ———, *Continuous analogues of matrix factorizations*, Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science, 471 (2015), p. 20140585.
- [33] L. N. TREFETHEN, *Householder triangularization of a quasimatrix*, IMA journal of numerical analysis, (2009), p. drp018.

Appendix A. Note on proof of Theorem 3.9.

Recall that we seek to bound the singular values of a sub-matrix $\sigma_{k+1}(\mathbf{F})$ of a vector-valued function by the singular values of the full vector-valued function $\sigma_{k+1}(F)$. The sub-matrix \mathbf{F} can be viewed as a discretized version of F .

This bound we need arises from the result in [20] where one considers a compact linear integral operator K with kernel $k(s, t)$ defined as

$$(A.1) \quad (Kf)(s) \equiv \int_{\Omega} k(s, t)f(t)dt, \quad s \in \Omega$$

In particular one can define a discretization of k using n values $[s_1, \dots, s_n]$ as \mathbf{K} such that $\mathbf{K}[i, j] = k(s_i, s_j)$. Then one can bound the eigenvalues of \mathbf{K} by the eigenvalues of K . The result is summarized in the lemma below.

LEMMA A.1 (Interlacing eigenvalues of discretized operator). *Let $K : L^2(\Omega) \rightarrow L^2(\Omega)$ be a symmetric, positive definite, compact linear integral operator with continuous kernel $k(s, t)$ that has eigendecomposition (from Mercer's theorem)*

$$k(s, t) = \sum_{p=1}^{\infty} \lambda_p \Psi_p(s) \Psi_p(t),$$

where $\{\lambda_p\}_{p=1}^{\infty}$ are eigenvalues and $\{\Psi_p(t)\}_{p=1}^{\infty}$ form an orthonormal basis for $L^2(\Omega)$. Furthermore, assume the eigenfunctions are uniformly bounded: $|\Psi_p(s)| < a$ for all p and s . Let $\mathbf{K} \in \mathbb{R}^{n \times n}$ denote a discretized version of k and let $\varepsilon > 1/2$. Then it holds that

$$(A.2) \quad \lambda_p(\mathbf{K}) \leq |\Omega| a^2 \zeta(2\varepsilon) k^{2\varepsilon} \lambda_p(k), \quad p < n$$

where ζ is the Riemann zeta function, $\zeta(2\varepsilon) = \sum_{p=1}^{\infty} \frac{1}{p^{2\varepsilon}}$.

If \mathbf{F} is the discretization of a continuous but non-symmetric kernel F , and the singular functions of the corresponding integral operator are uniformly bounded, then an analogous result for the singular values of \mathbf{F} can be shown to hold.